Contents

1 Linear Systems 1
   Matrix .................................................. 6
   ComplexMatrix ......................................... 11
   SparseMatrix ........................................... 16
   SparseMatrix.SparseArray ......................... 27
   ComplexSparseMatrix ............................. 28
   ComplexSparseMatrix.SparseArray ............ 37
   LU ..................................................... 39
   ComplexLU ........................................... 44
   SuperLU ............................................... 49
   SuperLU.ColumnOrdering ......................... 60
   SuperLU.PerformanceParameters ............. 61
   SuperLU.Scaling .................................. 62
   ComplexSuperLU .................................. 63
   ComplexSuperLU.ColumnOrdering ............ 74
   ComplexSuperLU.PerformanceParameters ... 75
   ComplexSuperLU.Scaling ....................... 76
   Cholesky ............................................. 77
   SparseCholesky .................................... 81
   SparseCholesky.NumericFactorization .... 87
   SparseCholesky.NumericFactor .............. 88
   SparseCholesky.SymbolicFactor .............. 88
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComplexSparseCholesky</td>
<td>88</td>
</tr>
<tr>
<td>ComplexSparseCholesky.NumericFactorization</td>
<td>94</td>
</tr>
<tr>
<td>ComplexSparseCholesky.NumericFactor</td>
<td>95</td>
</tr>
<tr>
<td>ComplexSparseCholesky.SymbolicFactor</td>
<td>95</td>
</tr>
<tr>
<td>QR</td>
<td>95</td>
</tr>
<tr>
<td>SVD</td>
<td>99</td>
</tr>
<tr>
<td>GenMinRes</td>
<td>104</td>
</tr>
<tr>
<td>GenMinRes.ImplementationMethod</td>
<td>118</td>
</tr>
<tr>
<td>GenMinRes.ResidualMethod</td>
<td>119</td>
</tr>
<tr>
<td>GenMinRes.IFunction</td>
<td>119</td>
</tr>
<tr>
<td>GenMinRes.IPreconditioner</td>
<td>120</td>
</tr>
<tr>
<td>GenMinRes.IVectorProducts</td>
<td>120</td>
</tr>
<tr>
<td>GenMinRes.INorm</td>
<td>121</td>
</tr>
<tr>
<td>ConjugateGradient</td>
<td>122</td>
</tr>
<tr>
<td>ConjugateGradient.IFunction</td>
<td>129</td>
</tr>
<tr>
<td>ConjugateGradient.IPreconditioner</td>
<td>130</td>
</tr>
</tbody>
</table>

2 Eigensystem Analysis

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eigen</td>
<td>131</td>
</tr>
<tr>
<td>SymEigen</td>
<td>132</td>
</tr>
</tbody>
</table>

3 Interpolation and Approximation

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spline</td>
<td>141</td>
</tr>
<tr>
<td>CsAkima</td>
<td>143</td>
</tr>
<tr>
<td>CsTCB</td>
<td>146</td>
</tr>
<tr>
<td>CsInterpolate</td>
<td>148</td>
</tr>
<tr>
<td>CsInterpolate.Condition</td>
<td>152</td>
</tr>
<tr>
<td>CsPeriodic</td>
<td>154</td>
</tr>
<tr>
<td>CsShape</td>
<td>155</td>
</tr>
<tr>
<td>CsSmooth</td>
<td>157</td>
</tr>
<tr>
<td>CsSmoothC2</td>
<td>158</td>
</tr>
<tr>
<td>CsTCB</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>163</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>BSpline</td>
<td>168</td>
</tr>
<tr>
<td>BsInterpolate</td>
<td>172</td>
</tr>
<tr>
<td>BsLeastSquares</td>
<td>174</td>
</tr>
<tr>
<td>Spline2D</td>
<td>177</td>
</tr>
<tr>
<td>Spline2DInterpolate</td>
<td>180</td>
</tr>
<tr>
<td>Spline2DLeastSquares</td>
<td>189</td>
</tr>
<tr>
<td>RadialBasis</td>
<td>195</td>
</tr>
<tr>
<td>RadialBasis.IFunction</td>
<td>207</td>
</tr>
<tr>
<td>RadialBasis.Gaussian</td>
<td>207</td>
</tr>
<tr>
<td>RadialBasis.HardyMultiquadric</td>
<td>209</td>
</tr>
</tbody>
</table>

4 Quadrature

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadrature</td>
<td>211</td>
</tr>
<tr>
<td>Quadrature.IFunction</td>
<td>212</td>
</tr>
<tr>
<td>HyperRectangleQuadrature</td>
<td>219</td>
</tr>
<tr>
<td>HyperRectangleQuadrature.IFunction</td>
<td>220</td>
</tr>
</tbody>
</table>

5 Differential Equations

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODE</td>
<td>225</td>
</tr>
<tr>
<td>ODE.ExamineStepOptions</td>
<td>227</td>
</tr>
<tr>
<td>ODE.ErrorNormOptions</td>
<td>231</td>
</tr>
<tr>
<td>OdeRungeKutta</td>
<td>231</td>
</tr>
<tr>
<td>OdeRungeKutta.IFunction</td>
<td>234</td>
</tr>
<tr>
<td>OdeAdamsGear</td>
<td>235</td>
</tr>
<tr>
<td>OdeAdamsGear.IntegrationType</td>
<td>240</td>
</tr>
<tr>
<td>OdeAdamsGear.SolveOption</td>
<td>240</td>
</tr>
<tr>
<td>OdeAdamsGear.IFunction</td>
<td>241</td>
</tr>
<tr>
<td>OdeAdamsGear.IJacobian</td>
<td>242</td>
</tr>
<tr>
<td>FeynmanKac</td>
<td>242</td>
</tr>
<tr>
<td>FeynmanKac.PDEStepControlMethod</td>
<td>297</td>
</tr>
<tr>
<td>FeynmanKac.IPdeCoefficients</td>
<td>298</td>
</tr>
<tr>
<td>FeynmanKac.IBoundaries</td>
<td>299</td>
</tr>
<tr>
<td>Function/Method</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>MinConGenLin.IGradient</td>
<td>399</td>
</tr>
<tr>
<td>BoundedLeastSquares</td>
<td>400</td>
</tr>
<tr>
<td>BoundedLeastSquares.IFunction</td>
<td>408</td>
</tr>
<tr>
<td>BoundedLeastSquares.IJacobian</td>
<td>409</td>
</tr>
<tr>
<td>BoundedVariableLeastSquares</td>
<td>409</td>
</tr>
<tr>
<td>NonNegativeLeastSquares</td>
<td>413</td>
</tr>
<tr>
<td>MinConNLP</td>
<td>417</td>
</tr>
<tr>
<td>MinConNLP.IFunction</td>
<td>429</td>
</tr>
<tr>
<td>MinConNLP.IGradient</td>
<td>430</td>
</tr>
<tr>
<td>NumericalDerivatives</td>
<td>431</td>
</tr>
<tr>
<td>NumericalDerivatives.IFunction</td>
<td>449</td>
</tr>
<tr>
<td>NumericalDerivatives.IJacobian</td>
<td>450</td>
</tr>
<tr>
<td>NumericalDerivatives.DifferencingMethod</td>
<td>451</td>
</tr>
</tbody>
</table>

9 Special Functions

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sfun</td>
<td>453</td>
</tr>
<tr>
<td>Bessel</td>
<td>469</td>
</tr>
</tbody>
</table>

10 Miscellaneous

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex</td>
<td>475</td>
</tr>
<tr>
<td>Physical</td>
<td>500</td>
</tr>
<tr>
<td>EpsilonAlgorithm</td>
<td>513</td>
</tr>
</tbody>
</table>

11 Printing Functions

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrintMatrix</td>
<td>517</td>
</tr>
<tr>
<td>PrintMatrixFormat</td>
<td>522</td>
</tr>
<tr>
<td>PrintMatrixFormat.ParsePosition</td>
<td>526</td>
</tr>
<tr>
<td>PrintMatrix.MatrixType</td>
<td>527</td>
</tr>
<tr>
<td>PrintMatrixFormat.FormatType</td>
<td>528</td>
</tr>
<tr>
<td>PrintMatrixFormat.ColumnLabelType</td>
<td>530</td>
</tr>
<tr>
<td>PrintMatrixFormat.RowLabelType</td>
<td>531</td>
</tr>
</tbody>
</table>

12 Basic Statistics

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents Contents</td>
<td>v</td>
</tr>
</tbody>
</table>
Summary ................................................................. 534
Covariances ............................................................ 545
Covariances.MatrixType ............................................ 551
PartialCovariances .................................................. 552
NormOneSample ...................................................... 559
NormTwoSample ...................................................... 565
Sort ................................................................. 576
Ranks .............................................................. 584
Ranks.Tie .......................................................... 592
EmpiricalQuantiles .................................................. 593
TableOneWay ......................................................... 596
TableTwoWay ........................................................ 601
TableMultiWay ......................................................... 608
TableMultiWay.TableBalanced .................................... 614
TableMultiWay.TableUnbalanced ................................ 615

13 Regression ......................................................... 617
RegressorsForGLM .................................................... 625
RegressorsForGLM.DummyType ................................... 633
LinearRegression ...................................................... 634
LinearRegression.CaseStatistics .................................. 643
LinearRegression.CoefficientTTestsValue ....................... 647
NonlinearRegression .................................................. 648
NonlinearRegression.IDerivative .................................. 662
NonlinearRegression.IFunction ................................... 663
SelectionRegression ................................................. 664
SelectionRegression.Criterion ..................................... 675
SelectionRegression.SummaryStatistics ......................... 675
StepwiseRegression ................................................... 677
StepwiseRegression.CoefficientTTestsValue .................... 687
StepwiseRegression.Direction ...................................... 688
UserBasisRegression .................................................. 689
IRegressionBasis ............................................. 694

14 Analysis of Variance .................................... 695
   ANCOVA .................................................. 695
   ANOVA .................................................... 708
   ANOVA.ComputeOption ................................... 718
   ANOVAFactorial ........................................... 719
   ANOVAFactorial.ErrorCalculation ....................... 729
   MultipleComparisons ................................... 730

15 Categorical and Discrete Data Analysis .............. 733
   ContingencyTable ....................................... 733
   CategoricalGenLinModel ................................ 747
   CategoricalGenLinModel.DistributionParameterModel 770

16 Nonparametric Statistics ................................. 773
   SignTest ............................................... 773
   WilcoxonRankSum ...................................... 777

17 Tests of Goodness of Fit ................................ 783
   ChiSquaredTest ........................................ 783
   NormalityTest .......................................... 788
   KolmogorovOneSample .................................. 793
   KolmogorovTwoSample .................................. 796

18 Time Series and Forecasting ............................. 801
   AutoCorrelation ........................................ 804
   AutoCorrelation.StdErr ................................ 813
   CrossCorrelation ........................................ 813
   CrossCorrelation.StdErr ................................ 826
   MultiCrossCorrelation .................................. 826
   LackOfFit ............................................... 838
   ARAutoUnivariate ....................................... 841
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARAutoUnivariate.ParamEstimation</td>
<td>862</td>
</tr>
<tr>
<td>ARSeasonalFit</td>
<td>862</td>
</tr>
<tr>
<td>ARSeasonalFit.CenterMethod</td>
<td>873</td>
</tr>
<tr>
<td>ARMA</td>
<td>874</td>
</tr>
<tr>
<td>ARMA.ParamEstimation</td>
<td>895</td>
</tr>
<tr>
<td>ARMAEstimateMissing</td>
<td>896</td>
</tr>
<tr>
<td>ARMAEstimateMissing.MissingValueEstimation</td>
<td>905</td>
</tr>
<tr>
<td>ARMAMaxLikelihood</td>
<td>906</td>
</tr>
<tr>
<td>ARMAOutlierIdentification</td>
<td>919</td>
</tr>
<tr>
<td>AutoARIMA</td>
<td>940</td>
</tr>
<tr>
<td>AutoARIMA.InformationCriterion</td>
<td>964</td>
</tr>
<tr>
<td>Difference</td>
<td>965</td>
</tr>
<tr>
<td>GARCH</td>
<td>970</td>
</tr>
<tr>
<td>KalmanFilter</td>
<td>976</td>
</tr>
</tbody>
</table>

19 Multivariate Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClusterKMeans</td>
<td>993</td>
</tr>
<tr>
<td>Dissimilarities</td>
<td>1002</td>
</tr>
<tr>
<td>Dissimilarities.Method</td>
<td>1007</td>
</tr>
<tr>
<td>Dissimilarities.Scaling</td>
<td>1009</td>
</tr>
<tr>
<td>ClusterHierarchical</td>
<td>1009</td>
</tr>
<tr>
<td>ClusterHierarchical.Linkage</td>
<td>1018</td>
</tr>
<tr>
<td>ClusterHierarchical.Transformation</td>
<td>1019</td>
</tr>
<tr>
<td>FactorAnalysis</td>
<td>1020</td>
</tr>
<tr>
<td>FactorAnalysis.MatrixType</td>
<td>1037</td>
</tr>
<tr>
<td>FactorAnalysis.Model</td>
<td>1037</td>
</tr>
<tr>
<td>DiscriminantAnalysis</td>
<td>1038</td>
</tr>
<tr>
<td>DiscriminantAnalysis.Discrimination</td>
<td>1060</td>
</tr>
<tr>
<td>DiscriminantAnalysis.CovarianceMatrix</td>
<td>1061</td>
</tr>
<tr>
<td>DiscriminantAnalysis.Classification</td>
<td>1061</td>
</tr>
<tr>
<td>DiscriminantAnalysis.PriorProbabilities</td>
<td>1062</td>
</tr>
</tbody>
</table>
### 20 Survival and Reliability Analysis

- KaplanMeierECDF .................................................. 1063
- KaplanMeierEstimates ........................................... 1067
- ProportionalHazards ............................................. 1075
- ProportionalHazards.TieHandling .............................. 1092
- LifeTables .......................................................... 1092

### 21 Probability Distribution Functions and Inverses

- Cdf ................................................................. 1101
- InvCdf ............................................................. 1137
- Pdf ................................................................. 1149
- ICdfFunction ....................................................... 1165
- InverseCdf ......................................................... 1165
- IDistribution ......................................................... 1167
- IProbabilityDistribution ........................................ 1168
- NormalDistribution ............................................. 1170
- GammaDistribution .............................................. 1172
- LogNormalDistribution .......................................... 1175
- PoissonDistribution ............................................. 1177

### 22 Random Number Generation

- Random ............................................................ 1182
- Random.BaseGenerator ......................................... 1206
- MersenneTwister .................................................. 1207
- MersenneTwister64 ............................................... 1211
- FaureSequence .................................................... 1216
- IRandomSequence ................................................. 1221

### 23 Finance

- Finance ............................................................ 1224
- Finance.Period .................................................... 1254
- Bond ............................................................... 1254
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bond.Frequency</td>
<td>1288</td>
</tr>
<tr>
<td>DayCountBasis</td>
<td>1289</td>
</tr>
<tr>
<td>IBasisPart</td>
<td>1291</td>
</tr>
<tr>
<td>Chart2D</td>
<td>1293</td>
</tr>
<tr>
<td>AbstractChartNode</td>
<td>1294</td>
</tr>
<tr>
<td>ChartNode</td>
<td>1310</td>
</tr>
<tr>
<td>Chart</td>
<td>1332</td>
</tr>
<tr>
<td>Background</td>
<td>1338</td>
</tr>
<tr>
<td>ChartTitle</td>
<td>1339</td>
</tr>
<tr>
<td>Grid</td>
<td>1340</td>
</tr>
<tr>
<td>Legend</td>
<td>1341</td>
</tr>
<tr>
<td>Annotation</td>
<td>1342</td>
</tr>
<tr>
<td>Axis</td>
<td>1346</td>
</tr>
<tr>
<td>AxisXY</td>
<td>1348</td>
</tr>
<tr>
<td>Axis1D</td>
<td>1353</td>
</tr>
<tr>
<td>AxisLabel</td>
<td>1358</td>
</tr>
<tr>
<td>AxisLine</td>
<td>1359</td>
</tr>
<tr>
<td>AxisTitle</td>
<td>1360</td>
</tr>
<tr>
<td>AxisUnit</td>
<td>1360</td>
</tr>
<tr>
<td>MajorTick</td>
<td>1361</td>
</tr>
<tr>
<td>MinorTick</td>
<td>1362</td>
</tr>
<tr>
<td>Transform</td>
<td>1362</td>
</tr>
<tr>
<td>TransformDate</td>
<td>1363</td>
</tr>
<tr>
<td>AxisR</td>
<td>1365</td>
</tr>
<tr>
<td>AxisRLabel</td>
<td>1367</td>
</tr>
<tr>
<td>AxisRLine</td>
<td>1368</td>
</tr>
<tr>
<td>AxisRMajorTick</td>
<td>1369</td>
</tr>
<tr>
<td>AxisTheta</td>
<td>1370</td>
</tr>
<tr>
<td>GridPolar</td>
<td>1372</td>
</tr>
<tr>
<td>Data</td>
<td>1372</td>
</tr>
<tr>
<td>ChartFunction</td>
<td>1383</td>
</tr>
</tbody>
</table>
ChartSpline ........................................ 1384
Text ............................................. 1385
ToolTip .......................................... 1388
FillPaint .......................................... 1390
Draw ............................................... 1393
FrameChart ...................................... 1401
PanelChart ....................................... 1403
DrawPick ......................................... 1405
PickEventArgs ................................. 1411
PickEventHandler .............................. 1413
WebChart ......................................... 1413
DrawMap .......................................... 1414
BoxPlot .......................................... 1420
BoxPlot.Statistics ............................ 1428
Contour .......................................... 1431
Contour.Legend ................................ 1444
ContourLevel .................................. 1445
ErrorBar ......................................... 1446
HighLowClose .................................. 1451
Candlestick ..................................... 1458
CandlestickItem ............................... 1460
SplineData ...................................... 1461
Bar ................................................ 1464
BarItem .......................................... 1473
BarSet .......................................... 1477
Pie ............................................... 1479
PieSlice .......................................... 1483
Dendrogram .................................... 1484
Polar ............................................. 1495
Heatmap ......................................... 1497
Heatmap.Legend ............................... 1509
Treemap .............................................................. 1510
Colormap ............................................................ 1517
Colormap_Fields ...................................................... 1518

25 Quality Control and Improvement Charts ........................ 1521
  ShewhartControlChart ........................................... 1521
  ControlLimit ...................................................... 1527
  XbarR .............................................................. 1529
  RChart ............................................................. 1536
  XbarS .............................................................. 1539
  SChart ............................................................. 1546
  XmR ................................................................. 1550
  NpChart ........................................................... 1553
  PChart ............................................................. 1556
  CChart ............................................................. 1560
  UChart ............................................................. 1563
  EWMA .............................................................. 1566
  CuSum .............................................................. 1569
  CuSumStatus ...................................................... 1572
  ParetoChart ........................................................ 1578

26 Data Mining ................................................................ 1585
  NaiveBayesClassifier ............................................. 1587

27 Neural Nets ................................................................ 1613
  FeedForwardNetwork ............................................. 1639
  Layer ................................................................. 1654
  InputLayer .......................................................... 1655
  HiddenLayer ......................................................... 1656
  OutputLayer ........................................................ 1657
  Node ................................................................. 1659
  InputNode ............................................................ 1660
<table>
<thead>
<tr>
<th>Class/Subclass</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceptron</td>
<td>1661</td>
</tr>
<tr>
<td>OutputPerceptron</td>
<td>1662</td>
</tr>
<tr>
<td>IActivation</td>
<td>1663</td>
</tr>
<tr>
<td>Activation</td>
<td>1664</td>
</tr>
<tr>
<td>Link</td>
<td>1665</td>
</tr>
<tr>
<td>ITrainer</td>
<td>1666</td>
</tr>
<tr>
<td>QuasiNewtonTrainer</td>
<td>1668</td>
</tr>
<tr>
<td>QuasiNewtonTrainer.IError</td>
<td>1674</td>
</tr>
<tr>
<td>LeastSquaresTrainer</td>
<td>1675</td>
</tr>
<tr>
<td>EpochTrainer</td>
<td>1680</td>
</tr>
<tr>
<td>BinaryClassification</td>
<td>1683</td>
</tr>
<tr>
<td>MultiClassification</td>
<td>1727</td>
</tr>
<tr>
<td>ScaleFilter</td>
<td>1742</td>
</tr>
<tr>
<td>ScaleFilter.ScalingMethod</td>
<td>1751</td>
</tr>
<tr>
<td>UnsupervisedNominalFilter</td>
<td>1752</td>
</tr>
<tr>
<td>UnsupervisedOrdinalFilter</td>
<td>1756</td>
</tr>
<tr>
<td>UnsupervisedOrdinalFilter.TransformMethod</td>
<td>1761</td>
</tr>
<tr>
<td>TimeSeriesFilter</td>
<td>1762</td>
</tr>
<tr>
<td>TimeSeriesClassFilter</td>
<td>1764</td>
</tr>
<tr>
<td>Network</td>
<td>1786</td>
</tr>
</tbody>
</table>

**28 Error Handling** 1795

<table>
<thead>
<tr>
<th>Class/Subclass</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning</td>
<td>1795</td>
</tr>
<tr>
<td>WarningObject</td>
<td>1797</td>
</tr>
<tr>
<td>IMSLException</td>
<td>1798</td>
</tr>
<tr>
<td>Logger</td>
<td>1799</td>
</tr>
</tbody>
</table>

**29 Exceptions** 1803

<table>
<thead>
<tr>
<th>Exception</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMSLUnexpectedErrorException</td>
<td>1806</td>
</tr>
<tr>
<td>AllConstraintsNotSatisfiedException</td>
<td>1806</td>
</tr>
<tr>
<td>BadInitialGuessException</td>
<td>1807</td>
</tr>
<tr>
<td>BoundaryInconsistentException</td>
<td>1808</td>
</tr>
<tr>
<td>Exception</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>BoundsInconsistentException</td>
<td>1810</td>
</tr>
<tr>
<td>ConstraintEvaluationException</td>
<td>1811</td>
</tr>
<tr>
<td>ConstraintsInconsistentException</td>
<td>1812</td>
</tr>
<tr>
<td>ConstraintsNotSatisfiedException</td>
<td>1814</td>
</tr>
<tr>
<td>CorrectorConvergenceException</td>
<td>1815</td>
</tr>
<tr>
<td>CyclingOccurringException</td>
<td>1816</td>
</tr>
<tr>
<td>DidNotConvergeException</td>
<td>1817</td>
</tr>
<tr>
<td>EqualityConstraintsException</td>
<td>1819</td>
</tr>
<tr>
<td>ErrorTestException</td>
<td>1820</td>
</tr>
<tr>
<td>FalseConvergenceException</td>
<td>1822</td>
</tr>
<tr>
<td>IllConditionedException</td>
<td>1823</td>
</tr>
<tr>
<td>InconsistentSystemException</td>
<td>1824</td>
</tr>
<tr>
<td>InitialConstraintsException</td>
<td>1825</td>
</tr>
<tr>
<td>InvalidMPSFileException</td>
<td>1827</td>
</tr>
<tr>
<td>IterationMatrixSingularException</td>
<td>1828</td>
</tr>
<tr>
<td>LimitingAccuracyException</td>
<td>1829</td>
</tr>
<tr>
<td>LinearlyDependentGradientsException</td>
<td>1830</td>
</tr>
<tr>
<td>MaxIterationsException</td>
<td>1831</td>
</tr>
<tr>
<td>MaxNumberStepsAllowedException</td>
<td>1833</td>
</tr>
<tr>
<td>MaxFcnEvalsExceededException</td>
<td>1834</td>
</tr>
<tr>
<td>MultipleSolutionsException</td>
<td>1835</td>
</tr>
<tr>
<td>NoAcceptablePivotException</td>
<td>1836</td>
</tr>
<tr>
<td>NoAcceptableStepsizeException</td>
<td>1837</td>
</tr>
<tr>
<td>NoConvergenceException</td>
<td>1839</td>
</tr>
<tr>
<td>NoLPSolutionException</td>
<td>1840</td>
</tr>
<tr>
<td>NoProgressException</td>
<td>1841</td>
</tr>
<tr>
<td>NotDefiniteAMatrixException</td>
<td>1842</td>
</tr>
<tr>
<td>NotDefiniteJacobiPreconditionerException</td>
<td>1843</td>
</tr>
<tr>
<td>NotDefinitePreconditionMatrixException</td>
<td>1845</td>
</tr>
<tr>
<td>NotSPDException</td>
<td>1846</td>
</tr>
<tr>
<td>NumericDifficultyException</td>
<td>1847</td>
</tr>
</tbody>
</table>
ObjectiveEvaluationException ........................................................... 1848
PenaltyFunctionPointInfeasibleException ........................................... 1849
ProblemInfeasibleException ............................................................. 1850
ProblemUnboundedException ......................................................... 1851
ProblemVacuousException .............................................................. 1852
QPInfeasibleException ................................................................. 1854
QPProblemUnboundedException ...................................................... 1855
SingularException ........................................................................... 1856
SingularMatrixException .................................................................. 1857
SingularPreconditionMatrixException .............................................. 1858
SolutionNotFoundException ............................................................... 1859
SomeConstraintsDiscardedException ............................................... 1860
TcurrentTstopInconsistentException .................................................. 1862
TEqualsToutException ...................................................................... 1863
TerminationCriteriaNotSatisfiedException ....................................... 1865
TimeIntervalTooSmallException ....................................................... 1866
ToleranceTooSmallException ............................................................ 1868
TooManyIterationsException ............................................................ 1869
TooManyStepsException .................................................................. 1871
TooMuchTimeException .................................................................... 1872
UnboundedBelowException ............................................................. 1873
VarBoundsInconsistentException ...................................................... 1875
WorkingSetSingularException ........................................................... 1876
AllDeletedException ........................................................................ 1877
AllMissingException ....................................................................... 1878
AltSeriesAccuracyLossException ..................................................... 1879
BadVarianceException ..................................................................... 1880
ClassificationVariableException ....................................................... 1882
ClassificationVariableLimitException .............................................. 1883
ClassificationVariableValueException ............................................. 1884
ClusterNoPointsException ............................................................... 1886
Chapter 1: Linear Systems

Types

class Matrix .......................................................... 6
class ComplexMatrix ................................................. 11
class SparseMatrix .................................................... 16
class SparseMatrix.SparseArray .................................. 27
class ComplexSparseMatrix .......................................... 28
class ComplexSparseMatrix.SparseArray ....................... 37
class LU ................................................................. 39
class ComplexLU ....................................................... 44
class SuperLU ........................................................ 49
enumeration SuperLU.ColumnOrdering ............................ 60
enumeration SuperLU.PerformanceParameters ................. 61
enumeration SuperLU.Scaling ...................................... 62
class ComplexSuperLU .............................................. 63
enumeration ComplexSuperLU.ColumnOrdering .................. 74
enumeration ComplexSuperLU.PerformanceParameters ....... 75
enumeration ComplexSuperLU.Scaling ........................... 76
class Cholesky ......................................................... 77
class SparseCholesky .................................................. 81
enumeration SparseCholesky.NumericFactorization .......... 87
class SparseCholesky.NumericFactor ............................ 88
class SparseCholesky.SymbolicFactor ......................... 88
class ComplexSparseCholesky ................................. 88
enumeration ComplexSparseCholesky.NumericFactorization 94
class ComplexSparseCholesky.NumericFactor ............... 95
class ComplexSparseCholesky.SymbolicFactor ............. 95
class QR ............................................................... 99
class GenMinRes .................................................... 104
enumeration GenMinRes.ImplementationMethod ............... 118
enumeration GenMinRes.ResidualMethod ........................ 119
interface GenMinRes.IFunction .................................. 119
**Usage Notes**

**Solving Systems of Linear Equations**

A square system of linear equations has the form \(Ax = b\), where \(A\) is a user-specified \(n \times n\) matrix, \(b\) is a given right-hand side \(n\) vector, and \(x\) is the solution \(n\) vector. Each entry of \(A\) and \(b\) must be specified by the user. The entire vector \(x\) is returned as output.

When \(A\) is invertible, a unique solution to \(Ax = b\) exists. The most commonly used direct method for solving \(Ax = b\) factors the matrix \(A\) into a product of triangular matrices and solves the resulting triangular systems of linear equations. Functions that use direct methods for solving systems of linear equations all compute the solution to \(Ax = b\).

**Matrix Factorizations**

In some applications, it is desirable to just factor the \(n \times n\) matrix \(A\) into a product of two triangular matrices. This can be done by a constructor of a class for solving the system of linear equations \(Ax = b\). The constructor of class \(LU\) computes the LU factorization of \(A\).

Besides the basic matrix factorizations, such as \(LU\) and \(LL^T\), additional matrix factorizations also are provided. For a real matrix \(A\), its QR factorization can be computed using the class QR. The class for computing the singular value decomposition (SVD) of a matrix is discussed in a later section.

**Matrix Inversions**

The inverse of an \(n \times n\) nonsingular matrix can be obtained by using the method Inverse in the classes for solving systems of linear equations. The inverse of a matrix need not be computed if the purpose is to solve one or more systems of linear equations. Even with multiple right-hand sides, solving a system of linear equations by computing the inverse and performing matrix multiplication is usually more expensive than the method discussed in the next section.

**Multiple Right-Hand Sides**

Consider the case where a system of linear equations has more than one right-hand side vector. It is most economical to find the solution vectors by first factoring the coefficient matrix \(A\) into products of triangular matrices. Then, the resulting triangular systems of linear equations are solved for each right-hand side. When \(A\) is a real general matrix, access to the LU factorization of \(A\) is computed by a constructor of LU. The solution \(x_k\) for the \(k\)-th right-hand side vector, \(b_k\) is then found by two triangular solves, \(Ly_k = b_k\) and \(Ux_k = y_k\). The method Solve in class LU is used to solve each right-hand side. These arguments are found in other functions for solving systems of linear equations.
**Least-Squares Solutions and QR Factorizations**

Least-squares solutions are usually computed for an over-determined system of linear equations $A_{m\times n}x = b$, where $m > n$. A least-squares solution $x$ minimizes the Euclidean length of the residual vector $r = Ax - b$. The class QR computes a unique least-squares solution for $x$ when $A$ has full column rank. If $A$ is rank-deficient, then the base solution for some variables is computed. These variables consist of the resulting columns after the interchanges. The QR decomposition, with column interchanges or pivoting, is computed such that $AP = QR$. Here, $Q$ is orthogonal, $R$ is upper-trapezoidal with its diagonal elements nonincreasing in magnitude, and $P$ is the permutation matrix determined by the pivoting. The base solution $x_B$ is obtained by solving $R(P^T)x = Q^Tb$ for the base variables. For details, see class QR. The QR factorization of a matrix $A$ such that $AP = QR$ with $P$ specified by the user can be computed using keywords.

**Singular Value Decompositions and Generalized Inverses**

The SVD of an $m \times n$ matrix $A$ is a matrix decomposition $A = USV^T$. With $q = \min(m,n)$, the factors $U_{m \times q}$ and $V_{n \times q}$ are orthogonal matrices, and $S_{q \times q}$ is a nonnegative diagonal matrix with nonincreasing diagonal terms. The class SVD computes the singular values of $A$ by default. Part or all of the $U$ and $V$ matrices, an estimate of the rank of $A$, and the generalized inverse of $A$, also can be obtained.

**Ill-Conditioning and Singularity**

An $m \times n$ matrix $A$, is mathematically singular if there is an $x \neq 0$ such that $Ax = 0$. In this case, the system of linear equations $Ax = b$ does not have a unique solution. On the other hand, a matrix $A$ is numerically singular if it is "close" to a mathematically singular matrix. Such problems are called ill-conditioned. If the numerical results with an ill-conditioned problem are unacceptable, users can obtain an approximate solution to the system using the SVD of $A$: If $q = \min(m,n)$ and

$$A = \sum_{i=1}^{q} s_i u_i v_i^T$$

then the approximate solution is given by the following:

$$x_k = \sum_{i=1}^{k} t_{i,i} \left(b^T u_i \right) v_i$$

The scalars $t_{i,i}$ are defined below.

$$t_{i,i} = \begin{cases} 
  s_{i,i}^{-1} & \text{if } s_{i,i} \geq \text{tol} > 0 \\
  0 & \text{otherwise}
\end{cases}$$

The user specifies the value of tol. This value determines how “close” the given matrix is to a singular matrix. Further restrictions may apply to the number of terms in the sum, $k \leq q$. For example, there may be a value of $k \leq q$ such that the scalars $\left|b^T u_i \right|, i > k$ are smaller than the average uncertainty in the right-hand side $b$. This means that these scalars can be replaced by zero; and hence, $b$ is replaced by a vector that is within the stated uncertainty of the problem.

**Sparse Matrix Storage Modes**

All classes that work with sparse matrices (e.g. classes SparseCholesky, SuperLU, ComplexSparseCholesky and ComplexSuperLU) require a matrix input format that only stores information about the nonzero entries of these matrices. IMSL C# Numerical Library supports two such formats: The sparse coordinate storage (SCS) format and the Sparse Array format. The SCS format stores the value of each matrix entry together with that entry’s row and column index. The Sparse Array
format stores the sparse matrix in two arrays of arrays. One array contains the references to the nonzero value arrays for each row of the matrix, the other contains the references to the associated column index arrays. As an example, consider the following sparse $6 \times 6$ matrix:

$$
A = \begin{pmatrix}
10 & 0 & 0 & 0 & 0 & 0 \\
0 & 10 & -3 & -1 & 0 & 0 \\
0 & 0 & 15 & 0 & 0 & 0 \\
-2 & 0 & 0 & 10 & -1 & 0 \\
-1 & 0 & 0 & -5 & 1 & -3 \\
-1 & -2 & 0 & 0 & 0 & 6 
\end{pmatrix}
$$

This matrix has 15 nonzero elements, and the sparse coordinate representation would be

| row | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 5 |
|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| col | 0 | 1 | 2 | 3 | 2 | 0 | 3 | 4 | 0 | 3 | 4 | 5 | 0 | 1 | 5 |
| val | 10 | -3 | -1 | 15 | -2 | 10 | -1 | -5 | 1 | -3 | 1 | -2 | 6 |

Since this representation does not rely on the order of the matrix elements, an equivalent form would be

| row | 5 | 4 | 3 | 0 | 5 | 1 | 2 | 1 | 4 | 3 | 1 | 4 | 3 | 5 | 4 |
|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| col | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 5 | 5 |
| val | -1 | -1 | -2 | 10 | -10 | 2 | 10 | 15 | -3 | 10 | -1 | 1 | -1 | 6 | -3 |

In Sparse Array format, matrix $A$ can be represented by the following two arrays, index and values:

**index:**
- row 0: 0
- row 1: 1, 2, 3
- row 2: 2
- row 3: 0, 3, 4
- row 4: 0, 3, 4, 5
- row 5: 0, 1, 5

**values:**
- row 0: 10
- row 1: 10, -3, -1
- row 2: 15
- row 3: -2, 10, -1
- row 4: -1, -5, 1, -3
- row 5: -1, -2, 6

In contrast to the SCS format, the row order is fixed here, but the entries within each row do not rely on any order.

In IMSL C# Numerical Library, sparse matrices are instances of classes `SparseMatrix/ComplexSparseMatrix`. Use of the SCS and Sparse Array input format is reflected by different constructors. The following code fragment shows how a user can construct a `SparseMatrix` for the matrix $A$ above in SCS format:

```csharp
// Generate an empty 6 by 6 SparseMatrix object
int nRows = nColumns = 6;
```
SparseMatrix a = new SparseMatrix(nRows, nColumns);
// Fill object a with the entries of matrix A,
// A given in SCS format
a.Set(0, 0, 10.0); a.Set(1, 1, 10.0); a.Set(1, 2, -3.0);
a.Set(1, 3, -1.0); a.Set(2, 2, 15.0); a.Set(3, 0, -2.0);
a.Set(3, 3, 10.0); a.Set(3, 4, -1.0); a.Set(4, 0, -1.0);
a.Set(4, 3, -5.0); a.Set(4, 4, 1.0); a.Set(4, 5, -3.0);
a.Set(5, 0, -1.0); a.Set(5, 1, -2.0); a.Set(5, 5, 6.0);

Similarly, the following code fragment shows how to construct a SparseMatrix in Sparse Array format:

```csharp
// Generate matrix A in Sparse Array format
int nRows = nColumns = 6;
int[][] aIndex = {{0},
                  {1, 2, 3},
                  {2},
                  {0, 3, 4},
                  {0, 3, 4, 5},
                  {0, 1, 5}};

int[][] aValues = {{10.0},
                   {10.0, -3.0, -1.0},
                   {15.0},
                   {-2.0, 10.0, -1.0},
                   {-1.0, -5.0, 1.0, -3.0},
                   {-1.0, -2.0, 6.0}};

// Create SparseMatrix object based on A in Sparse Array format
SparseMatrix a = new SparseMatrix(nRows, nColumns, aIndex, aValues);
```

All IMSL C# Numerical Library algorithms working on sparse matrices use SparseMatrix or ComplexSparseMatrix as an input argument type. A square $n \times n$ matrix is called symmetric if $A^T = A$, where $A^T$ denotes the transpose of $A$. It is called Hermitian if $A^H = A$. Here, $A^H \equiv \bar{A}^T$ denotes the conjugate transpose of $A$. For these types of matrices, all matrix information is stored in the lower triangular part of the matrix. Therefore, for all IMSL C# Numerical Library sparse matrix classes working on symmetric or Hermitian matrices it is sufficient to store only the lower triangle of the matrix in the SparseMatrix or ComplexSparseMatrix input object.

The complex $4 \times 4$ matrix

$$
H = \begin{pmatrix}
4 & 1-i & 0 & 0 \\
1+i & 4 & 1-i & 0 \\
0 & 1+i & 4 & 1-i \\
0 & 0 & 1+i & 4
\end{pmatrix}
$$

is Hermitian positive definite. Typically, matrices of this type are input to methods of class ComplexSparseCholesky.

The following code fragment shows how the lower triangular part of $H$ can be constructed as a ComplexSparseMatrix type in SCS input format:

```csharp
// Generate an empty 4 by 4 ComplexSparseMatrix object
int nRows = nColumns = 4;
ComplexSparseMatrix h = new ComplexSparseMatrix(nRows, nColumns);
// Store lower triangular part of H in object h
```
Matrix Class

public class Imsl.Math.Matrix

Matrix manipulation functions.

Methods

Add
static public double[,] Add(double[,] a, double[,] b)

Description
Add two rectangular matrices, a + b.

Parameters
  a – A double rectangular matrix.
  b – A double rectangular matrix.

Returns
A double rectangular matrix representing the matrix sum of the two arguments.

Exception
System.ArgumentException is thrown when the matrices are not the same size

CheckSquareMatrix
static public void CheckSquareMatrix(double[,] a)

Description
Check that the matrix is square.

Parameter
  a – A double matrix.

Exception
System.ArgumentException is thrown when the matrix is not square

FrobeniusNorm
static public double FrobeniusNorm(double[,] a)
Description
Return the frobenius norm of a matrix.

Parameter
   a – A double rectangular matrix.

Returns
A double scalar value equal to the frobenius norm of the matrix.

InfinityNorm
static public double InfinityNorm(double[,] a)

Description
Return the infinity norm of a matrix.

Parameter
   a – A double rectangular matrix.

Returns
A double scalar value equal to the maximum of the row sums of the absolute values of the matrix elements.

InverseLowerTriangular
static public double[,] InverseLowerTriangular(double[,] a)

Description
Returns the inverse of the lower triangular matrix a.

Parameter
   a – A double square lower triangular matrix.

Returns
A double matrix containing the inverse of a.

InverseUpperTriangular
static public double[,] InverseUpperTriangular(double[,] a)

Description
Returns the inverse of the upper triangular matrix a.

Parameter
   a – A double square upper triangular matrix.

Returns
A double matrix containing the inverse of a.

Multiply
static public double[] Multiply(double[] x, double[,] a)

Linear Systems
Description
Return the product of the row matrix x and the rectangular matrix a.

Parameters
- x – A double row matrix.
- a – A double rectangular matrix.

Returns
A double vector representing the product of the arguments, x*a.

Exception
System.ArgumentException is thrown when the number of elements in the input row matrix is not equal to the number of rows of the matrix

Multiply
static public double[] Multiply(double[] x, double[,] a, int processors)

Description
Return the product of the row matrix x and the rectangular matrix a.

Parameters
- x – A double row matrix.
- a – A double rectangular matrix.
- processors – An int which specifies the number of processors to use. If processors is less than 1, then processors = 1 is used.

Returns
A double vector representing the product of the arguments, x*a.

Exception
System.ArgumentException is thrown when the number of elements in the input row matrix is not equal to the number of rows of the matrix

Multiply
static public double[] Multiply(double[,] a, double[] x)

Description
Multiply the rectangular matrix a and the column matrix x.

Parameters
- a – A double rectangular matrix.
- x – A double column matrix.

Returns
A double vector representing the product of the arguments, a * x.
Exception
System.ArgumentException is thrown when the number of columns in the input matrix is not equal to the number of elements in the input column vector.

Multiply
static public double[,] Multiply(double[,] a, double[,] b, int processors)

Description
Multiply two rectangular arrays, a * b, using multiple processors.

Parameters
- a – A double rectangular array.
- b – A double rectangular array.
- processors – An int which specifies the number of processors to use. If processors is less than 1, then processors = 1 is used.

Returns
The double matrix product of a times b.

Multiply
static public double[,] Multiply(double[,] a, double[,] b)

Description
Multiply two rectangular matrices, a * b.

Parameters
- a – A double rectangular matrix.
- b – A double rectangular matrix.

Returns
The double matrix product of a * b.

Exception
System.ArgumentException is thrown when the number of columns in a is not equal to the number of rows in b.

OneNorm
static public double OneNorm(double[,] a)

Description
Return the matrix one norm.

Parameter
- a – A double rectangular matrix.
Returns
A double value equal to the maximum of the column sums of the absolute values of the matrix elements.

Subtract
static public double[,] Subtract(double[,] a, double[,] b)

Description
Subtract two rectangular matrixs, a - b.

Parameters
a – A double rectangular matrix.
b – A double rectangular matrix.

Returns
A double rectangular matrix representing the matrix difference of the two arguments.

Exception
SystemArgumentException is thrown when the matrices are not the same size

Transpose
static public double[,] Transpose(double[,] a)

Description
Return the transpose of a matrix.

Parameter
a – A double matrix.

Returns
A double matrix which is the transpose of the argument.

Example: Matrix and PrintMatrix
The 1 norm of a matrix is found using a method from the Matrix class. The matrix is printed using the PrintMatrix class.

```csharp
using System;
using Imsl.Math;

public class MatrixEx1
{
    public static void Main(String[] args)
    {
        double nrm1;
        double[,] a = {
            {0.0, 1.0, 2.0, 3.0},
            {4.0, 5.0, 6.0, 7.0},
            {8.0, 9.0, 8.0, 1.0},
            {6.0, 3.0, 4.0, 3.0}
        }
```
\begin{verbatim}
Get the 1 norm of matrix a
nrm1 = Matrix.OneNorm(a);

Construct a PrintMatrix object with a title
PrintMatrix p = new PrintMatrix("A Simple Matrix");

Print the matrix and its 1 norm
p.Print(a);
Console.Out.WriteLine("The 1 norm of the matrix is " + nrm1);
\end{verbatim}

Output

\begin{verbatim}
A Simple Matrix
0 1 2 3
0 0 1 2 3
1 4 5 6 7
2 8 9 8 1
3 6 3 4 3
\end{verbatim}

The 1 norm of the matrix is 20

\section*{ComplexMatrix Class}

public class Imsl.Math.ComplexMatrix
Complex matrix manipulation functions.

\section*{Methods}

\subsection*{Add}
static public Imsl.Math.Complex[,] Add(Imsl.Math.Complex[,] a,
Imsl.Math.Complex[,] b)

\textbf{Description}
Add two rectangular Complex arrays, \(a + b\).

\textbf{Parameters}
- \(a\) – A Complex rectangular array.
- \(b\) – A Complex rectangular array.
Returns
The Complex matrix sum of the two arguments.

Exception
System.ArgumentException is thrown when the matrices are not the same size.

CheckSquareMatrix
static public void CheckSquareMatrix(Imsl.Math.Complex[,] a)

Description
Check that the Complex matrix is square.

Parameter
a – A Complex matrix.

Exception
System.ArgumentException is thrown when the matrix is not square.

FrobeniusNorm
static public double FrobeniusNorm(Imsl.Math.Complex[,] a)

Description
Return the frobenius norm of a Complex matrix.

Parameter
a – A Complex rectangular matrix.

Returns
A double value equal to the frobenius norm of the matrix.

InfinityNorm
static public double InfinityNorm(Imsl.Math.Complex[,] a)

Description
Return the infinity norm of a Complex matrix.

Parameter
a – A Complex rectangular matrix.

Returns
A double value equal to the maximum of the row sums of the absolute values of the array elements.

Multiply

Description
Returns the product of the row vector x and the rectangular array a, both Complex.
Parameters
   x – A Complex row vector.
   a – A Complex rectangular matrix.

Returns
A Complex vector containing the product of the arguments, \( x \ast a \).

Exception
   System.ArgumentException is thrown when the number of elements in the input vector is not equal to the number of rows of the matrix.

Multiply

Description
Returns the product of the row vector \( x \) and the rectangular array \( a \), both Complex.

Parameters
   x – A Complex row vector.
   a – A Complex rectangular matrix.
   processors – An int which specifies the number of processors to use. If processors is less than 1, then processors = 1 is used.

Returns
A Complex vector containing the product of the arguments, \( x \ast a \).

Exception
   System.ArgumentException is thrown when the number of elements in the input vector is not equal to the number of rows of the matrix.

Multiply

Description
Multiply the rectangular array \( a \) and the column vector \( x \), both Complex.

Parameters
   a – A Complex rectangular matrix.
   x – A Complex vector.

Returns
A Complex vector containing the product of the arguments, \( a \ast x \).
**Exception**

`System.ArgumentException` is thrown when the number of columns in the input matrix is not equal to the number of elements in the input vector.

---

**Multiply**

```csharp
```

**Description**

Multiply two Complex rectangular arrays, $a \times b$.

**Parameters**

- $a$ – A Complex rectangular array.
- $b$ – A Complex rectangular array.

**Returns**

The Complex matrix product of $a \times b$.

---

**Multiply**

```csharp
```

**Description**

Multiply two Complex rectangular arrays, $a \times b$, using multiple processors.

**Parameters**

- $a$ – A Complex rectangular array.
- $b$ – A Complex rectangular array.
- processors – An `int` which specifies the number of processors to use. If `processors` is less than 1, then `processors = 1` is used.

**Returns**

The Complex matrix product of $a \times b$.

---

**OneNorm**

```csharp
static public double OneNorm(Imsl.Math.Complex[,] a)
```

**Description**

Return the Complex matrix one norm.

**Parameter**

- $a$ – A Complex rectangular array.
Returns
A double value equal to the maximum of the column sums of the absolute values of the array elements.

Subtract

Description
Subtract two Complex rectangular arrays, a - b.

Parameters
a – A Complex rectangular array.
b – A Complex rectangular array.

Returns
The Complex matrix difference of the two arguments.

Exception
System.ArgumentException is thrown when the matrices are not the same size.

Transpose

Description
Return the transpose of a Complex matrix.

Parameter
a – A Complex matrix.

Returns
The Complex matrix transpose of the argument.

Example: Print a Complex Matrix
A Complex matrix and its transpose is printed.

using System;
using Imsl.Math;

class ComplexMatrixEx1
{
    public static void Main(String[] args)
    {
        Complex[,] a = {
            {new Complex(1, 3), new Complex(3, 5), new Complex(7, 9)},
            {new Complex(8, 7), new Complex(9, 5), new Complex(1, 9)},
            {new Complex(2, 9), new Complex(6, 9), new Complex(7, 3)},
            {new Complex(5, 4), new Complex(8, 4), new Complex(5, 9)}
        };
    }
// Print the matrix
new PrintMatrix("Matrix A").Print(a);

// Print the transpose of the matrix
new PrintMatrix("Transpose(A)").Print(ComplexMatrix.Transpose(a));

Output

<table>
<thead>
<tr>
<th>Matrix A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
</tr>
<tr>
<td>0 1+3i 3+5i 7+9i</td>
</tr>
<tr>
<td>1 8+7i 9+5i 1+9i</td>
</tr>
<tr>
<td>2 2+9i 6+9i 7+3i</td>
</tr>
<tr>
<td>3 5+4i 8+4i 5+9i</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transpose(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3</td>
</tr>
<tr>
<td>0 1+3i 8+7i 2+9i 5+4i</td>
</tr>
<tr>
<td>1 3+5i 9+5i 6+9i 8+4i</td>
</tr>
<tr>
<td>2 7+9i 1+9i 7+3i 5+9i</td>
</tr>
</tbody>
</table>

SparseMatrix Class

public class Imsl.Math.SparseMatrix

A general real sparse matrix intended to be efficiently and easily updated.

A SparseMatrix can be constructed from a set of arrays, or it can be abstractly created as an empty array and then incrementally built into final form. It is usually easier to create an empty SparseMatrix of a set size and then use the Set method to set the elements of the array. When setting the elements of the sparse array, their positions should be thought of as their positions in the dense array. Elements can be set in any order, but only the elements actually set are stored.

This class includes methods to update the sparse matrix. There are also methods to multiply a sparse matrix and a vector or to multiply two sparse matrices. To solve a sparse linear system use SparseCholesky or SuperLU.

See Also

Imsl.Math.SparseCholesky (p. 81), Imsl.Math.SuperLU (p. 49)
Properties

**NumberOfColumns**

public int NumberOfColumns {get; }

**Description**
The number of columns in the matrix.

**Property Value**
An int containing the number of columns in the matrix.

**NumberOfNonZeros**

public long NumberOfNonZeros {get; }

**Description**
The number of nonzeros in the matrix.

**Property Value**
A long containing the number of nonzeros in the matrix.

**NumberOfRows**

public int NumberOfRows {get; }

**Description**
The number of rows in the matrix.

**Property Value**
An int containing the number of rows in the matrix.

Constructors

**SparseMatrix**

public SparseMatrix(int nRows, int nColumns)

**Description**
Creates a new instance of SparseMatrix. Initially this is the zero matrix.

**Parameters**

- nRows – An int specifying the number of rows in the sparse matrix.
- nColumns – An int specifying the number of columns in the sparse matrix.

**SparseMatrix**

public SparseMatrix(Imsl.Math.SparseMatrix A)
Description
Creates a new instance of SparseMatrix which is a copy of another SparseMatrix object.

Parameter
A – A SparseMatrix object containing the matrix to be copied.

SparseMatrix

Description
Constructs a sparse matrix from a SparseArray.

Parameter
sparseArray – A SparseArray used to initialize the sparse matrix. The field NumberOfNonZeros in SparseArray (p. 27) is not used for initialization, so it does not have to be set.

SparseMatrix
public SparseMatrix(int nRows, int nColumns, int[][] index, double[][] values)

Description
Constructs a sparse matrix from SparseArray data.

Parameters
nRows – An int specifying the number of rows in the sparse matrix.
nColumns – An int specifying the number of columns in the sparse matrix.
index – An int jagged array containing the column indices of all nonzero elements corresponding to the compressed representation of the sparse matrix in values. The size of index must be identical to the size of values. The i-th row contains the column indices of all nonzero elements of row i of the sparse matrix. The j-th element of row i is the column index of the value located at the same position in values.
values – A double jagged array containing the compressed representation of a real sparse matrix of size nRows by nColumns. The number of rows in values must be nRows. The i-th row contains all nonzero elements of row i of the full sparse matrix.

Methods

Add

Description
Performs element-wise addition of two real sparse matrices A, B of type SparseMatrix, \( C \leftarrow \alpha A + \beta B \).
Parameters
alpha – A double scalar value applied to SparseMatrix A.
beta – A double scalar value applied to SparseMatrix B.
A – A SparseMatrix matrix.
B – A SparseMatrix matrix.

Returns
A SparseMatrix matrix representing the computed sum.

Exception
System.ArgumentException is thrown when the matrices are not of the same size.

CheckSquareMatrix
public void CheckSquareMatrix()

Description
Check that the matrix is square.

Exception
System.ArgumentException is thrown if the matrix is not square.

FrobeniusNorm
public double FrobeniusNorm()

Description
Returns the Frobenius norm of the matrix.

Returns
A double scalar equal to the Frobenius norm of the matrix.

Get
public double Get(int iRow, int jColumn)

Description
Returns the value of an element in the matrix.

Parameters
iRow – An int specifying the row index of the element.
jColumn – An int specifying the column index of the element.

Returns
A double containing the value of the iRow-th and jColumn-th element. If the element was never set, its value is zero.
Description
Returns the infinity norm of the matrix.

Returns
A double scalar equal to the maximum of the row sums of the absolute values of the array elements of the sparse matrix.

Multiply

public double[] Multiply(double[] x)

Description
Multiply the matrix by a vector.

Parameter
x – A double column array.

Returns
A double vector representing the product of the constructed matrix and x, Ax.

Exception
System.ArgumentException is thrown when the number of columns of the matrix represented by this object is not equal to the number of elements in the input column vector.

Multiply

static public double[] Multiply(Imsl.Math.SparseMatrix A, double[] x)

Description
Multiply sparse matrix A and column array x, Ax.

Parameters
A – A SparseMatrix matrix.
x – A double column array.

Returns
A double vector representing the product of the arguments, Ax.

Exception
System.ArgumentException is thrown when the number of columns in the input matrix is not equal to the number of elements in the input column vector.

Multiply

static public double[] Multiply(double[] x, Imsl.Math.SparseMatrix A)

Description
Multiply row array x and sparse matrix A, x^T A.
Parameters
- **x** – A double row array.
- **A** – A `SparseMatrix` matrix.

Returns
A double vector representing the product of the arguments, $x^TA$.

Exception
- `System.ArgumentException` is thrown when the number of elements in the input vector is not equal to the number of rows of the matrix.

### Multiply
```csharp
```

Description
Multiply two sparse matrices, $C \leftarrow AB$.

Parameters
- **A** – A `SparseMatrix` sparse matrix.
- **B** – A `SparseMatrix` sparse matrix.

Returns
The `SparseMatrix` product $AB$ of $A$ and $B$.

Exception
- `System.ArgumentException` is thrown when the number of columns of matrix $A$ is not equal to the number of rows of matrix $B$.

### MultiplySymmetric
```csharp
static public double[] MultiplySymmetric(Imsl.Math.SparseMatrix A, double[] x)
```

Description
Multiply sparse symmetric matrix $A$ and column vector $x$.

Parameters
- **A** – A `SparseMatrix` sparse symmetric matrix, where only the lower triangular part of the matrix is to be used.
- **x** – A double vector.

Returns
A double vector representing the product of the arguments, $Ax$.

Exception
- `System.ArgumentException` is thrown when the input matrix is not square or the number of columns in the input matrix is not equal to the number of elements in the input column vector.

### OneNorm
```csharp
public double OneNorm()
```

---

**Linear Systems**

**SparseMatrix** • 21
Description
Returns the matrix one norm of the sparse matrix.

Returns
A double scalar containing the maximum of the column sums of the absolute values of the array elements.

PlusEquals
public double PlusEquals(int iRow, int jColumn, double x)

Description
Adds a value to an element in the matrix.

Parameters
- iRow – An int specifying the row index of the element.
- jColumn – An int specifying the column index of the element.
- x – A double specifying the value to be added to the iRow-th and jColumn-th element.

Returns
A double containing the updated value of the iRow-th and jColumn-th element, which equals its old value plus \( x \).

Set
public void Set(int iRow, int jColumn, double x)

Description
Sets the value of an element in the matrix.

Parameters
- iRow – An int specifying the row index of the element.
- jColumn – An int specifying the column index of the element.
- x – A double specifying the value of the iRow-th and jColumn-th element.

ToDenseMatrix
public double[,] ToDenseMatrix()

Description
Returns the sparse matrix as a dense matrix.

Returns
A double rectangular array containing this matrix with all of the zeros explicitly present. The number of rows and columns in the returned matrix is the same as in the sparse matrix.

ToSparseArray
public Imsl.Math.SparseMatrix.SparseArray ToSparseArray()
Description

Returns the sparse matrix in the SparseArray form.

Returns

A SparseArray object representing the sparse matrix.

See Also: Imsl.Math.SparseMatrix.SparseArray (p. 27)

Transpose

public Imsl.Math.SparseMatrix Transpose()

Description

Returns the transpose of the matrix.

Returns

A SparseMatrix matrix containing the transpose of the constructed SparseMatrix object.

Example 1: SparseMatrix

The matrix product of two matrices is computed using a method from the SparseMatrix class. The one norm of the result is also computed. The matrix and its norm are printed.

```java
using System;
using Imsl.Math;

public class SparseMatrixEx1
{
    public static void Main(String[] args)
    {
        SparseMatrix b = new SparseMatrix(6, 6);
        b.Set(0, 0, 10.0);
        b.Set(1, 1, 10.0);
        b.Set(1, 2, -3.0);
        b.Set(1, 3, -1.0);
        b.Set(2, 2, 15.0);
        b.Set(3, 0, -2.0);
        b.Set(3, 3, 10.0);
        b.Set(3, 4, -1.0);
        b.Set(4, 0, -1.0);
        b.Set(4, 3, -5.0);
        b.Set(4, 4, 1.0);
        b.Set(5, 0, -1.0);
        b.Set(5, 1, -2.0);
        b.Set(5, 5, 6.0);

        SparseMatrix c = new SparseMatrix(6, 3);
        c.Set(0, 0, 5.0);
        c.Set(1, 2, -3.0);
        c.Set(2, 0, 1.0);
        c.Set(2, 2, 7.0);
        c.Set(3, 0, 2.0);
        c.Set(4, 1, -5.0);
```
c.Set(4, 2, 2.0);
c.Set(5, 2, 4.0);

SparseMatrix A = SparseMatrix.Multiply(b, c);

// Get the one norm of matrix A
Console.Out.WriteLine("The 1-norm of the matrix is "+ A.OneNorm());
SparseMatrix.SparseArray sa = A.ToSparseArray();

// Print the matrix and its one norm
Console.Out.WriteLine("row column value");
for (int i = 0; i < sa.NumberOfRows; i++)
{
    for (int j = 0; j < sa.Index[i].Length; j++)
    {
        int jj = sa.Index[i][j];
        Console.Out.WriteLine(" "+ i + " "+ j + " "+ sa.Values[i][j]);
    }
}

Output
The 1-norm of the matrix is 198
row column value
0 0 50
1 0 -5
1 2 -51
2 0 15
2 2 105
3 0 10
3 1 5
3 2 -2
4 0 -15
4 1 -5
4 2 -10
5 0 -5
5 2 30

Example 2: SparseMatrix Using Matrix Market Format

The matrix market exchange format is an ASCII file format that represents sparse matrices in coordinate format. It consists of three sections: The header section is the first line in the file and contains general informations about the matrix, e.g. data type and symmetry properties. This line is followed by the comments section which consists of zero or more lines of comments. The remainder of the file is the data section. Its first line contains the row number, column number and number of nonzeros of the matrix. The following lines contain location and value of all nonzero entries of the matrix, usually one per line. A file in Matrix Market format is read and converted to a SparseMatrix. Matrix information and its one norm are printed.
using System;
using Imsl.Math;

class SparseMatrixEx2
{
    class MTXReader
    {
        public string TypeCode
        {
            get
            {
                return this.typecode;
            }
        }
        public SparseMatrix Matrix
        {
            get
            {
                return this.matrix;
            }
        }
        private string typecode;
        private SparseMatrix matrix;
        public virtual void read(string filename)
        {
            System.IO.FileStream aFile = System.IO.File.OpenRead(filename);
            System.IO.StreamReader br = new System.IO.StreamReader(aFile);

            // read type code initial line
            string line = br.ReadLine();
            typecode = line.Replace('%%', ' ');

            // read comment lines if any
            bool comment = true;
            while (comment)
            {
                line = br.ReadLine();
                comment = line.StartsWith('%');
            }

            // line now contains the size information which needs to be parsed
            string[] str = line.Split(new char[] { ' ' });
            int nRows = (int)Parse(str[0].Trim());
            int nColumns = (int)Parse(str[1].Trim());
            int nNonZeros = (int)Parse(str[2].Trim());

            // now we're into the data section
            matrix = new SparseMatrix(nRows, nColumns);
            while (true)
            {
                line = br.ReadLine();
                if (line == null)
                {
                    break;
                }
            }
        }
    }
}
str = line.Split(new Char[] {' '});
int i = (Int32.Parse(str[0].Trim()));
int j = (Int32.Parse(str[1].Trim()));

double x=0.0;
if (str[2].Trim() == "")
{
    x = Convert.ToDouble(str[3].Trim());
}
else
{
    x = Convert.ToDouble(str[2].Trim());
}
matrix.Set(i - 1, j - 1, x);

public static void Main(String[] args)
{
    MTXReader mr = new MTXReader();
mr.read("bcsstk01.mtx");
SparseMatrix A = mr.Matrix;

    // Print the matrix type
Console.Out.WriteLine("The matrix type is " + mr.TypeCode);

    // Print the matrix information and its one norm
Console.Out.WriteLine("The number of rows is " + A.NumberOfRows);
int ncols = A.NumberOfColumns;
Console.Out.WriteLine("The number of columns is " + ncols);
long nnz = A.NumberOfNonZeros;
Console.Out.WriteLine("The number of nonzero elements is " + nnz);
Console.Out.WriteLine();
Console.Out.WriteLine("The 1 norm of the matrix is " + A.OneNorm());
}

Output

The matrix type is MatrixMarket matrix coordinate real symmetric
The number of rows is 48
The number of columns is 48
The number of nonzero elements is 224

The 1 norm of the matrix is 3009444444.44744
SparseMatrix.SparseArray Class

public class Imsl.Math.SparseMatrix.SparseArray

The SparseArray class uses public fields to hold the data for a sparse matrix in the Sparse Array format.

This format came about as a means for storing sparse matrices. In this format, a sparse matrix is represented by two arrays of arrays. One array contains the references to the nonzero value arrays for each row of the sparse matrix. The other contains the references to the associated column index arrays.

As an example, consider the following real sparse matrix:

\[
A = \begin{bmatrix}
10.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 10.0 & -3.0 & -1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 15.0 & 0.0 & 0.0 & 0.0 \\
-2.0 & 0.0 & 0.0 & 10.0 & -1.0 & 0.0 \\
-1.0 & 0.0 & 0.0 & -5.0 & 1.0 & -3.0 \\
-1.0 & -2.0 & 0.0 & 0.0 & 0.0 & 6.0 \\
\end{bmatrix}
\]

In SparseArray, this matrix can be represented by the two jagged arrays, values and index, where values refers to the nonzero entries in \(A\) and index to the column indices:

```java
double values[][] = {{10.0},
                    {10.0, -3.0, -1.0},
                    {15.0},
                    {-2.0, 10.0, -1.0},
                    {-1.0, -5.0, 1.0, -3.0},
                    {-1.0, -2.0, 6.0}};

int index[][] = {{0},
                 {1, 2, 3},
                 {2},
                 {0, 3, 4},
                 {0, 3, 4, 5},
                 {0, 1, 5}};
```

Fields

Index

public int[][] Index
Description
Jagged array containing column indices. The length of this array equals numberOfRows. The length of each row equals the number of nonzeros in that row of the sparse matrix.

**NumberOfColumns**
public int NumberOfColumns

Description
Number of columns in the matrix.

**NumberOfNonZeros**
public long NumberOfNonZeros

Description
Number of nonzeros in the matrix.

**NumberOfRows**
public int NumberOfRows

Description
Number of rows in the matrix.

**Values**
public double[][] Values

Description
Jagged array containing sparse array values. This array must have the same shape as Index.

**Constructor**

**SparseArray**
public SparseArray()

Description

**ComplexSparseMatrix Class**

public class Imsl.Math.ComplexSparseMatrix

A general complex sparse matrix which is intended to be efficiently and easily updated.
A ComplexSparseMatrix can be constructed from a set of arrays, or it can be abstractly created as an empty array and then incrementally built into final form. It is usually easier to create an empty ComplexSparseMatrix of set size and then use the Set method to set the elements of the array. When setting the elements of the sparse array, their positions should be thought of as their positions in the dense array. Elements can be set in any order, but only the elements set are stored.

This class includes methods to update the sparse matrix. There are also methods to multiply a sparse matrix and a vector or to multiply two sparse matrices. To solve a sparse linear system use ComplexSparseCholesky or ComplexSuperLU.

See Also

ComplexSparseCholesky (p. 88), ComplexSuperLU (p. 63)

Properties

**NumberOfColumns**

public int NumberOfColumns {get; }

Description

The number of columns in the matrix.

Property Value

An int containing the number of columns in the matrix.

**NumberOfNonZeros**

public long NumberOfNonZeros {get; }

Description

The number of nonzeros in the matrix.

Property Value

A long containing the number of nonzeros in the matrix.

**NumberOfRows**

public int NumberOfRows {get; }

Description

The number of rows in the matrix.

Property Value

An int containing the number of rows in the matrix.
Constructors

ComplexSparseMatrix
public ComplexSparseMatrix(int nRows, int nColumns)

Description
Creates a new instance of ComplexSparseMatrix. Initially this is the zero matrix.

Parameters
- nRows – An int containing the number of rows in the sparse matrix.
- nColumns – An int containing the number of columns in the sparse matrix.

ComplexSparseMatrix
public ComplexSparseMatrix(Imsl.Math.ComplexSparseMatrix A)

Description
Creates a new instance of ComplexSparseMatrix which is a copy of another ComplexSparseMatrix object.

Parameter
- A – A ComplexSparseMatrix object containing the complex sparse matrix to be copied.

ComplexSparseMatrix

Description
Constructs a complex sparse matrix from a SparseArray object.

Parameter
- sparseArray – A ComplexSparseMatrix.SparseArray used to initialize the sparse matrix. The field NumberOfNonZeros in SparseArray (p. 37) is not used for initialization, therefore does not have to be set.

ComplexSparseMatrix
public ComplexSparseMatrix(int nRows, int nColumns, int[][] index, Imsl.Math.Complex[][] values)

Description
Constructs a sparse matrix from SparseArray data.

Parameters
- nRows – An int containing the number of rows in the sparse matrix.
- nColumns – An int containing the number of columns in the sparse matrix.
index – An int jagged array containing the column indices of all nonzero elements corresponding to the compressed representation of the sparse matrix in values. The size of index must be identical to the size of values. The i-th row contains the column indices of all nonzero elements of row i of the sparse matrix. The j-th element of row i is the column index of the value located at the same position in values.

values – A Complex jagged array containing the compressed representation of a complex sparse matrix of size nRows by nColumns. The number of rows in values must be nRows. The i-th row contains all nonzero elements of row i of the full sparse matrix.

Methods

Add

Description
Performs element-wise addition of two complex sparse matrices A, B of type ComplexSparseMatrix, 
\[ C \leftarrow \alpha A + \beta B. \]

Parameters
alpha – A Complex scalar value applied to ComplexSparseMatrix A.
beta – A Complex scalar value applied to ComplexSparseMatrix B.
A – A ComplexSparseMatrix matrix.
B – A ComplexSparseMatrix matrix.

Returns
A ComplexSparseMatrix matrix representing the computed sum.

Exception
System.ArgumentException is thrown when the matrices are not of the same size.

CheckSquareMatrix
public void CheckSquareMatrix()

Description
Check that the matrix is square.

Exception
System.ArgumentException is thrown if the matrix is not square.

ConjugateTranspose
public Imsl.Math.ComplexSparseMatrix ConjugateTranspose()
Description
Returns the conjugate transpose of the matrix.

Returns
A ComplexSparseMatrix object which is the conjugate transpose of the constructed ComplexSparseMatrix.

FrobeniusNorm
public double FrobeniusNorm()

Description
Returns the Frobenius norm of the matrix.

Returns
A double containing the Frobenius norm of the matrix.

Get
public Imsl.Math.Complex Get(int iRow, int jColumn)

Description
Returns the value of an element in the matrix.

Parameters
iRow – An int containing the row index of the element.
jColumn – An int containing the column index of the element.

Returns
A Complex containing the value of the iRow-th and jColumn-th element. If the element was never set, its value is zero.

InfinityNorm
public double InfinityNorm()

Description
Returns the infinity norm of the matrix.

Returns
A double containing the maximum of the row sums of the absolute values of the array elements of the sparse matrix.

Multiply

Description
Multiply the matrix by a vector.

Parameter
x – A Complex column array.
Returns
A Complex vector representing the product of the constructed matrix and $x$, $Ax$.

Exception
System.ArgumentException is thrown when the number of columns of the matrix represented by this object is not equal to the number of elements in the input column vector.

Multiply

Description
Multiply sparse matrix $A$ and column array $x$, $Ax$.

Parameters
$A$ – A ComplexSparseMatrix matrix.
$x$ – A Complex column array.

Returns
A Complex vector representing the product of the arguments, $Ax$.

Exception
System.ArgumentException is thrown when the number of columns in the input matrix is not equal to the number of elements in the input column vector.

Multiply

Description
Multiply row array $x$ and sparse matrix $A$, $x^TA$.

Parameters
$x$ – A Complex row array.
$A$ – A ComplexSparseMatrix matrix.

Returns
A Complex vector representing the product of the arguments, $x^TA$.

Exception
System.ArgumentException is thrown when the number of elements in the input vector is not equal to the number of rows of the matrix.

Multiply
Multiply two sparse matrices, $C \leftarrow AB$.

**Parameters**
- $A$ – A `ComplexSparseMatrix` sparse matrix.
- $B$ – A `ComplexSparseMatrix` sparse matrix.

**Returns**
The `ComplexSparseMatrix` product $AB$ of $A$ and $B$.

**Exception**
`System.ArgumentException` is thrown when the column number of matrix $A$ is not equal to the row number of matrix $B$.

---

**MultiplyHermitian**

**Description**
Multiply sparse Hermitian matrix $A$ and column vector $x$.

**Parameters**
- $A$ – A Hermitian `ComplexSparseMatrix`, where only the lower triangular part of the matrix is to be used.
- $x$ – A `Complex` vector.

**Returns**
A `Complex` vector representing the product of the arguments, $Ax$.

**Exception**
`System.ArgumentException` is thrown when the input matrix is not square or the number of columns in the input matrix is not equal to the number of elements in the input column vector.

---

**OneNorm**
public `double` OneNorm()

**Description**
Returns the matrix one norm of the sparse matrix.

**Returns**
A `double` containing the maximum of the column sums of the absolute values of the array elements.

---

**PlusEquals**

**Description**
Adds a value to an element in the matrix.
**Parameters**

- `iRow` – An `int` containing the row index of the element.
- `jColumn` – An `int` containing the column index of the element.
- `x` – A `Complex` containing the value to be added to the `iRow`-th and `jColumn`-th element.

**Returns**

A `Complex` containing the updated value of the `iRow`-th and `jColumn`-th element, which equals its old value plus `x`.

---

### Set

```java
public void Set(int iRow, int jColumn, Imsl.Math.Complex x)
```

**Description**

Sets the value of an element in the matrix.

**Parameters**

- `iRow` – An `int` containing the row index of the element.
- `jColumn` – An `int` containing the column index of the element.
- `x` – A `Complex` containing the value of the `iRow`-th and `jColumn`-th element.

---

### ToDenseMatrix

```java
public Imsl.Math.Complex[,] ToDenseMatrix()
```

**Description**

Returns the sparse matrix as a dense matrix.

**Returns**

A `Complex` rectangular matrix containing this matrix with all of the zeros explicitly present. The number of rows and columns in the returned matrix is the same as in the sparse matrix.

---

### ToSparseArray

```java
```

**Description**

Returns the sparse matrix in the `SparseArray` form.

**Returns**

A `ComplexSparseMatrix.SparseArray` containing the complex sparse matrix in sparse array form.

See Also: Imsl.Math.ComplexSparseMatrix.SparseArray (p. 37)

---

### Example: ComplexSparseMatrix

The matrix product of two complex sparse matrices is computed using a method from the `ComplexSparseMatrix` class. The one-norm of the result is also computed. The matrix and its norm are printed.
using System;
using Imsl.Math;

public class ComplexSparseMatrixEx1
{
    public static void Main(String[] args)
    {
        ComplexSparseMatrix b = new ComplexSparseMatrix(6, 6);
b.Set(0, 0, new Complex(10.0, -3.0));
b.Set(1, 1, new Complex(10.0, 0.0));
b.Set(1, 2, new Complex(-3.0, 2.0));
b.Set(1, 3, new Complex(-1.0, -1.0));
b.Set(2, 2, new Complex(15.0, 5.0));
b.Set(3, 0, new Complex(-2.0, 0.0));
b.Set(3, 3, new Complex(10.0, 1.0));
b.Set(3, 4, new Complex(-1.0, -2.0));
b.Set(4, 0, new Complex(-2.0, 1.0));
b.Set(4, 3, new Complex(-5.0, 7.0));
b.Set(4, 4, new Complex(1.0, -3.0));
b.Set(4, 5, new Complex(-3.0, 0.0));
b.Set(5, 0, new Complex(-1.0, 4.0));
b.Set(5, 1, new Complex(-2.0, 1.0));
b.Set(5, 5, new Complex(6.0, -5.0));

        ComplexSparseMatrix c = new ComplexSparseMatrix(6, 3);
c.Set(0, 0, new Complex(5.0, 0.0));
c.Set(1, 2, new Complex(-3.0, -4.0));
c.Set(2, 0, new Complex(1.0, 2.0));
c.Set(2, 2, new Complex(7.0, 1.0));
c.Set(3, 0, new Complex(2.0, -7.0));
c.Set(4, 1, new Complex(-5.0, 2.0));
c.Set(4, 2, new Complex(2.0, 1.0));
c.Set(5, 2, new Complex(4.0, 0.0));

        // A = b * c
        ComplexSparseMatrix A = ComplexSparseMatrix.Multiply(b, c);

        // Get the one norm of matrix A
        Console.Out.WriteLine("The 1-norm of the matrix is " + A.OneNorm());

        ComplexSparseMatrix.SparseArray sa = A.ToSparseArray();

        Console.Out.WriteLine(" row column value");
        for (int i = 0; i < sa.NumberOfRows; i++)
        {
            for (int j = 0; j < sa.Index[i].Length; j++)
            {
            }
        }
    }
}
The 1-norm of the matrix is 253.937005114344

\[
\begin{array}{ccc}
\text{row} & \text{column} & \text{value} \\
0 & 0 & (50,-15) \\
1 & 0 & (-16,1) \\
1 & 2 & (-53,-29) \\
2 & 0 & (5,35) \\
2 & 2 & (100,50) \\
3 & 0 & (17,-68) \\
3 & 1 & (9,8) \\
3 & 2 & (0,-5) \\
4 & 0 & (34,49) \\
4 & 1 & (1,17) \\
4 & 2 & (-7,-5) \\
5 & 0 & (-5,20) \\
5 & 2 & (34,-15) \\
\end{array}
\]

### ComplexSparseMatrix.SparseArray Class

public class Imsl.Math.ComplexSparseMatrix.SparseArray

The SparseArray class uses public fields to hold the data for a sparse matrix in the Sparse Array format. This format came about as a means for storing sparse matrices. In this format, a sparse matrix is represented by two arrays of arrays. One array contains the references to the nonzero value arrays for each row of the sparse matrix. The other contains the references to the associated column index arrays. As an example, consider the following complex sparse matrix:

\[
A = \begin{pmatrix}
10.0-3.0i & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 10.0 & -3.0+2.0i & -1.0-1.0i & 0.0 & 0.0 \\
0.0 & 0.0 & 15.0+5.0i & 0.0 & 0.0 & 0.0 \\
-2.0 & 0.0 & 0.0 & 10.0+1.0i & -1.0-2.0i & 0.0 \\
-1.0 & 0.0 & 0.0 & -5.0+7.0i & 1.0-3.0i & -3.0 \\
-1.0+4.0i & -2.0+1.0i & 0.0 & 0.0 & 0.0 & 6.0-5.0i
\end{pmatrix}
\]

In SparseArray, this matrix can be represented by the two jagged arrays, values and index, where values refers to the nonzero entries in A and index to the column indices:

```java
Complex values[][] = [
{new Complex(10.0, -3.0)},
{new Complex(10.0, 0.0), new Complex(-3.0, 2.0),
new Complex(-1.0,-1.0)},
{new Complex(15.0, 5.0)},
{new Complex(-2.0, 0.0), new Complex(10.0, 1.0),
new Complex(-1.0, -2.0)},
{new Complex(-1.0, 0.0), new Complex(-5.0, 7.0),
new Complex(1.0, -3.0), new Complex(-3.0, 0.0)},
{new Complex(-1.0, 4.0), new Complex(-2.0, 1.0), new Complex(6.0, -5.0)} ]; int
index[][] = { {0}, {1, 2, 3}, {2}, {0, 3, 4}, {0, 3, 4, 5}, {0, 1, 5} };
```
Fields

Index
public int[][] Index

Description
Jagged array containing column indices. The length of this array equals numberOfRows. The length of each row equals the number of nonzeros in that row of the sparse matrix.

NumberOfColumns
public int NumberOfColumns

Description
Number of columns in the matrix.

NumberOfNonZeros
public long NumberOfNonZeros

Description
Number of nonzeros in the matrix.

NumberOfRows
public int NumberOfRows

Description
Number of rows in the matrix.

Values
public Imsl.Math.Complex[][] Values

Description
Jagged array containing sparse array values. This array must have the same shape as Index.

Constructor

SparseArray
public SparseArray()

Description
LU Class

```java
class Imsl.Math.LU {
    LU factorization of a matrix of type double.
    
    LU performs an LU factorization of a real general coefficient matrix. The Condition method estimates
    the reciprocal of the $L_1$ condition number of the matrix. The LU factorization is done using scaled partial
    pivoting. Scaled partial pivoting differs from partial pivoting in that the pivoting strategy is the same as if
    each row were scaled to have the same infinity norm.
    
    The $L_1$ condition number of the matrix $A$ is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to
    compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used
    by LINPACK and is described in a paper by Cline et al. (1979).
    
    Note that $A$ is not retained for use by other methods of this class, only the factorization of $A$ is retained.
    Thus, $A$ is a required parameter to the condition method.
    
    An estimated condition number greater than $1/\varepsilon$ (where $\varepsilon$ is machine precision) indicates that very small
    changes in $A$ can cause very large changes in the solution $x$. Iterative refinement can sometimes find the
    solution to such a system. If there is concern about the input matrix being ill-conditioned, the user of this
    class should check the condition number of the input matrix using the condition method before using
    one of the other class methods.
    
    LU fails if $U$, the upper triangular part of the factorization, has a zero diagonal element. This can occur
    only if $A$ either is singular or is very close to a singular matrix.
    
    Use the Solve method to solve systems of equations. The Determinant method can be called to
    compute the determinant of the coefficient matrix.
    
    LU is based on the LINPACK routine SGECO; see Dongarra et al. (1979). SGECO uses unscaled partial
    pivoting.
}
```

Property

**NumberOfProcessors**

```java
public int NumberOfProcessors {get; set; }
```

**Description**

Perform the parallel calculations with the maximum possible number of processors set to
NumberOfProcessors.

**Property Value**

An int indicating the maximum possible number of processors to use.

Linear Systems
Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

Constructor

LU
public LU(double[,] a)

Description
Creates the LU factorization of a square matrix of type double.

Parameter
a – The double square matrix to be factored.

Methods

Condition
public double Condition(double[,] a)

Description
Return an estimate of the reciprocal of the $L_1$ condition number of a matrix.

Parameter
a – The double square matrix for which the reciprocal of the $L_1$ condition number is desired.

Returns
A double value representing an estimate of the reciprocal of the $L_1$ condition number of the matrix.

Determinant
public double Determinant()

Description
Return the determinant of the matrix used to construct this instance.

Returns
A double scalar containing the determinant of the matrix used to construct this instance.

GetL
public double[,] GetL()
**Description**
Returns the lower triangular portion of the $LU$ factorization of $A$.

**Returns**
A double matrix containing $L$, the lower triangular portion of the $LU$ factorization of $A$.

**Remarks**
Scaled partial pivoting is used to achieve the $LU$ factorization. The resulting factorization is such that $AP = LU$, where $A$ is the input matrix, $P$ is the permutation matrix returned by GetPermutationMatrix, $L$ is the lower triangular matrix returned by GetL, and $U$ is the unit upper triangular matrix returned by GetU.

### GetPermutationMatrix
**public double[,] GetPermutationMatrix()**

**Description**
Returns the permutation matrix which results from the $LU$ factorization of $A$.

**Returns**
A double matrix containing the permuted identity matrix as a result of the $LU$ factorization of $A$.

**Remarks**
Scaled partial pivoting is used to achieve the $LU$ factorization. The resulting factorization is such that $AP = LU$, where $A$ is the input matrix, $P$ is the permutation matrix returned by GetPermutationMatrix, $L$ is the lower triangular matrix returned by GetL, and $U$ is the unit upper triangular matrix returned by GetU.

### GetU
**public double[,] GetU()**

**Description**
Returns the unit upper triangular portion of the $LU$ factorization of $A$.

**Returns**
A double matrix containing $U$, the unit upper triangular portion of the $LU$ factorization of $A$.

**Remarks**
Scaled partial pivoting is used to achieve the $LU$ factorization. The resulting factorization is such that $AP = LU$, where $A$ is the input matrix, $P$ is the permutation matrix returned by GetPermutationMatrix, $L$ is the lower triangular matrix returned by GetL, and $U$ is the unit upper triangular matrix returned by GetU.

### Inverse
**public double[,] Inverse()**

**Description**
Returns the inverse of the matrix used to construct this instance.
Returns
A double matrix representing the inverse of the matrix used to construct this instance.

Solve
public double[] Solve(double[] b)

Description
Solve $Ax = b$ for $x$ using the $LU$ factorization of $A$, where $A$ is the input matrix $a$.

Parameter
b – A double array containing the right-hand side of the linear system.

Returns
A double array containing the solution to the linear system of equations.

Solve
static public double[] Solve(double[,] a, double[] b)

Description
Solve $Ax = b$ for $x$ using the $LU$ factorization of $A$.

Parameters
a – A double square matrix.
b – A double column vector.

Returns
A double column vector containing the solution to the linear system of equations.

Exceptions
System.ArgumentException is thrown when the number of rows in the input matrix is not equal to the number of elements in $x$
Imsl.Math.SingularMatrixException is thrown when the matrix is singular

SolveTranspose
public double[] SolveTranspose(double[] b)

Description
Return the solution $x$ of the linear system $A^T = b$.

Parameter
b – A double array containing the right-hand side of the linear system.

Returns
A double array containing the solution to the linear system of equations.
Example: LU Factorization of a Matrix

The LU Factorization of a Matrix is performed. A linear system is then solved using the factorization. The inverse, determinant, and condition number of the input matrix are also computed.

```csharp
using System;
using Imsl.Math;

public class LUEx1
{
    public static void Main(String[] args)
    {
        double[,] a = {
            {1, 3, 3},
            {1, 3, 4},
            {1, 4, 3}
        };
        double[] b = new double[] { 12, 13, 14 };

        // Compute the LU factorization of A
        LU lu = new LU(a);

        // Solve Ax = b
        double[] x = lu.Solve(b);
        new PrintMatrix("x").Print(x);

        // Find the inverse of A.
        double[,] ainv = lu.Inverse();
        new PrintMatrix("ainv").Print(ainv);

        // Find the condition number of A.
        double condition = lu.Condition(a);
        Console.Out.WriteLine("condition number = " + condition);

        // Find the determinant of A.
        double determinant = lu.Determinant();
        Console.Out.WriteLine("determinant = " + determinant);
    }
}
```

Output

```
x
0 0 3
1 2 1

ainv
0 1 2
0 7 -3 -3
1 -1 2.22044604925031E-16 1
2 -1 1 0
```

Linear Systems
condition number = 0.0151202749140893

determinant = -1

---

**ComplexLU Class**

```csharp
public class Imsl.Math.ComplexLU

LU factorization of a matrix of type Complex.

ComplexLU performs an LU factorization of a complex general coefficient matrix. ComplexLU’s method Condition estimates the condition number of the matrix. The LU factorization is done using scaled partial pivoting. Scaled partial pivoting differs from partial pivoting in that the pivoting strategy is the same as if each row were scaled to have the same infinity norm.

The $L_1$ condition number of the matrix $A$ is defined to be $\kappa (A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

Note that $A$ is not retained for use by other methods of this class, only the factorization of $A$ is retained. Thus, $A$ is a required parameter to the condition method.

An estimated condition number greater than $1/\varepsilon$ (where $\varepsilon$ is machine precision) indicates that very small changes in $A$ can cause very large changes in the solution $x$. Iterative refinement can sometimes find the solution to such a system.

ComplexLU fails if $U$, the upper triangular part of the factorization, has a zero diagonal element. This can occur only if $A$ either is singular or is very close to a singular matrix.

The Solve method can be used to solve systems of equations. The method Determinant can be called to compute the determinant of the coefficient matrix.

ComplexLU is based on the LINPACK routine CGECO; see Dongarra et al. (1979). CGECO uses unscaled partial pivoting.

---

**Property**

**NumberOfProcessors**

```csharp
public int NumberOfProcessors { get; set; }
```

**Description**

Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.
Property Value
An int indicating the maximum possible number of processors to use.

Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

Constructor

**ComplexLU**

```csharp
public ComplexLU(Imsl.Math.Complex[,] a)
```

Description
Creates the LU factorization of a square matrix of type Complex.

Parameter
- `a` – The Complex square matrix to be factored.

Exception
- `Imsl.Math.SingularMatrixException` is thrown when the input matrix is singular

Methods

**Condition**

```csharp
public double Condition(Imsl.Math.Complex[,] a)
```

Description
Return an estimate of the reciprocal of the $L_1$ condition number.

Parameter
- `a` – A Complex matrix.

Returns
A double scalar value representing the estimate of the reciprocal of the $L_1$ condition number of the matrix $A$, where $A$ is a.

**Determinant**

```csharp
public Imsl.Math.Complex Determinant()
```

Description
Return the determinant of the matrix used to construct this instance.
**Returns**
A Complex scalar containing the determinant of the matrix used to construct this instance.

**GetL**
public Imsl.Math.Complex[,] GetL()

**Description**
Returns the lower triangular portion of the LU factorization of A.

**Returns**
A Complex matrix containing L, the lower triangular portion of the LU factorization of A.

**Remarks**
Scaled partial pivoting is used to achieve the LU factorization. The resulting factorization is such that AP = LU, where A is the input matrix a, P is the permutation matrix returned by GetPermutationMatrix, L is the lower triangular matrix returned by GetL, and U is the unit upper triangular matrix returned by GetU.

**GetPermutationMatrix**
public Imsl.Math.Complex[,] GetPermutationMatrix()

**Description**
Returns the permutation matrix which results from the LU factorization of A.

**Returns**
A double matrix containing the permuted identity matrix as a result of the LU factorization of A.

**Remarks**
Scaled partial pivoting is used to achieve the LU factorization. The resulting factorization is such that AP = LU, where A is the input matrix a, P is the permutation matrix returned by GetPermutationMatrix, L is the lower triangular matrix returned by GetL, and U is the unit upper triangular matrix returned by GetU.

**GetU**
public Imsl.Math.Complex[,] GetU()

**Description**
Returns the unit upper triangular portion of the LU factorization of A.

**Returns**
A Complex matrix containing U, the unit upper triangular portion of the LU factorization of A.

**Remarks**
Scaled partial pivoting is used to achieve the LU factorization. The resulting factorization is such that AP = LU, where A is the input matrix a, P is the permutation matrix returned by GetPermutationMatrix, L is the lower triangular matrix returned by GetL, and U is the unit upper triangular matrix returned by GetU.

**Inverse**
public Imsl.Math.Complex[,] Inverse()
**Description**

Returns the inverse of the matrix used to construct this instance.

**Returns**

A `Complex` matrix containing the inverse of the matrix used to construct this object.

---

**Solve**

```csharp
```

**Description**

Return the solution $x$ of the linear system $Ax = b$ using the LU factorization of $A$, where $A$ is the input matrix $a$.

**Parameter**

- $b$ – A `Complex` array containing the right-hand side of the linear system.

**Returns**

A `Complex` array containing the solution to the linear system of equations.

---

**Solve**

```csharp
```

**Description**

Return the solution $x$ of the linear system $Ax = b$ using the LU factorization of $A$.

**Parameters**

- $a$ – A `Complex` square matrix.
- $b$ – A `Complex` column vector.

**Returns**

A `Complex` column vector containing the solution to the linear system of equations.

**Exceptions**

- `System.ArgumentException` is thrown when the number of rows in $A$ is not equal to the length of $b$
- `Imsl.Math.SingularMatrixException` is thrown when the matrix is singular

---

**SolveTranspose**

```csharp
```

**Description**

Return the solution $x$ of the linear system $A^T x = b$.

**Parameter**

- $b$ – A `Complex` array containing the right-hand side of the linear system.

---

**Linear Systems**
Returns
A Complex array containing the solution to the linear system of equations.

Example: LU Decomposition of a Complex Matrix

The Complex structure is used to convert a real matrix to a Complex matrix. An LU decomposition of the matrix is performed. The reciprocal of the condition number of the Matrix is then computed and checked against machine precision to determine whether or not to issue a Warning about the results. A linear system is then solved using the factorization. The determinant of the input matrix is also computed.

class ComplexLUEx1
{
    public static void Main(String[] args)
    {
        double[,] ar = {{1, 3, 3},
                        {1, 3, 4},
                        {1, 4, 3}};
        double[] br = {12, 13, 14};
        Complex[,] a = new Complex[3,3];
        Complex[] b = new Complex[3];
        for (int i = 0; i < 3; i++)
        {
            b[i] = new Complex(br[i]);
            for (int j = 0; j < 3; j++)
            {
                a[i,j] = new Complex(ar[i,j]);
            }
        }
        // Compute the LU factorization of A
        ComplexLU clu = new ComplexLU(a);
        // Check the reciprocal of the condition number of A
        // against machine precision
        double condition = clu.Condition(a);
        if(condition <= 2.220446049250313e-16)
        {
            Console.Out.WriteLine("WARNING. The matrix is too ill-conditioned.");
            Console.Out.WriteLine("An estimate of the reciprocal of its L1 "+
                                 "condition number is "+condition+".");
            Console.Out.WriteLine("Results based on this factorization may "+
                                 "not be accurate.");
        }
        // Solve Ax = b
        Complex[] x = clu.Solve(b);
        Console.Out.WriteLine("The solution is:");
        new PrintMatrix("x").Print(x);
    }
}
// Print the condition number of A.
Console.Out.WriteLine("The condition number = " + condition);
Console.Out.WriteLine();

// Find the determinant of A.
Complex determinant = clu.Determinant();
Console.Out.WriteLine("The determinant = " + determinant);

Output
The solution is:

\[
x
\]
0 3
1 2
2 1

The condition number = 0.0148867313915858
The determinant = -0.99999999999999978

SuperLU Class

public class Imsl.Math.SuperLU

Computes the LU factorization of a general sparse matrix of type SparseMatrix by a column method
and solves the real sparse linear system of equations \( Ax = b \).

Consider the sparse linear system of equations

\[
Ax = b.
\]

Here, \( A \) is a general square, nonsingular, \( n \) by \( n \) sparse matrix, and \( x \) and \( b \) are vectors of length \( n \). All
entries in \( A \), \( x \) and \( b \) are of type \( \text{double} \).

Gaussian elimination, applied to the system above, can be shortly described as follows:
1. Compute a triangular factorization \( P_D A \sqrt{D} P_c = LU \). Here, \( D \) and \( D_r \) are positive definite diagonal
matrices to equilibrate the system and \( P_l \) and \( P_c \) are permutation matrices to ensure numerical stability
and preserve sparsity. \( L \) is a unit lower triangular matrix and \( U \) is an upper triangular matrix.
2. Solve \( Ax = b \) by evaluating

\[
x = A^{-1} b = D_r (P_l (U^{-1} (L^{-1} (P_c (D_r b)))).
\]
This is done efficiently by multiplying from right to left in the last expression: Scale the rows of \( b \) by \( D_r \). Multiplying \( P_r(D_r b) \) means permuting the rows of \( D_r b \). Multiplying \( L^{-1}(P_r(D_r b)) \) means solving the triangular system of equations with matrix \( L \) by substitution. Similarly, multiplying \( U^{-1}(L^{-1}(P_r(D_r b)) \) means solving the triangular system with \( U \).

Class SuperLU handles step 1 above in the Solve method if it has not been computed prior to step 2. More precisely, before \( Ax = b \) is solved the following steps are performed:

1. Equilibrate matrix \( A \), i.e. compute diagonal matrices \( D_r \) and \( D_c \) so that \( \hat{A} = D_r A D_c \) is “better conditioned” than \( A \), i.e. \( \hat{A}^{-1} \) is less sensitive to perturbations in \( \hat{A} \) than \( A^{-1} \) is to perturbations in \( A \).

2. Order the columns of \( \hat{A} \) to increase the sparsity of the computed \( L \) and \( U \) factors, i.e. replace \( \hat{A} \) by \( \hat{A} P_c \) where \( P_c \) is a column permutation matrix.

3. Compute the \( LU \) factorization of \( \hat{A} P_c \). For numerical stability, the rows of \( \hat{A} P_c \) are eventually permuted through the factorization process by scaled partial pivoting, leading to the decomposition \( \hat{A} := P_c \hat{A} P := LU \). The \( LU \) factorization is done by a left looking supernode-panel algorithm with 2-D blocking. See Demmel, Eisenstat, Gilbert et al. (1999) for further information on this technique.

4. Compute the reciprocal pivot growth factor

\[
\max_{1 \leq j \leq n} \frac{\|\hat{A}_j\|_\infty}{\|U_j\|_\infty}
\]

where \( \hat{A}_j \) and \( U_j \) denote the \( j \)-th column of matrices \( \hat{A} \) and \( U \), respectively.

5. Estimate the reciprocal of the condition number of matrix \( \hat{A} \).

Method Solve uses this information to perform the following steps:

1. Solve the system \( Ax = b \) using the computed triangular factors.

2. Iteratively refine the solution, again using the computed triangular factors. This is equivalent to Newton’s method.

3. Compute forward and backward error bounds for the solution vector \( x \).

Some of the steps mentioned above are optional. Their settings can be controlled by the Set methods and properties of class SuperLU.

Class SuperLU is based on the SuperLU code written by Demmel, Gilbert, Li et al. For more detailed explanations of the factorization and solve steps, see the SuperLU Users’ Guide (1999).

Copyright (c) 2003, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from U.S. Dept. of Energy).

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

50 • SuperLU

IMSL C# Numerical Library
(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

(2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

(3) Neither the name of Lawrence Berkeley National Laboratory, U.S. Dept. of Energy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Properties

ColumnOrderingMethod

```csharp
public Imsl.Math.SuperLU.ColumnOrdering ColumnOrderingMethod {get; set; }
```

Description

The method used to permute the columns of the input matrix.

Property Value

A ColumnOrdering scalar specifying how the columns of the input matrix are to be permuted for sparsity preservation.

<table>
<thead>
<tr>
<th>value</th>
<th>method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural</td>
<td>natural ordering, that is $P_c = I$, $I$ the identity matrix.</td>
</tr>
<tr>
<td>MinimumDegreeAtPlusA</td>
<td>minimum degree ordering on the structure of $A^T + A$</td>
</tr>
<tr>
<td>MinimumDegreeAtA</td>
<td>minimum degree ordering on the structure of $A^T A$</td>
</tr>
<tr>
<td>ColumnApproximateMinimumDegree</td>
<td>column approximate minimum degree ordering</td>
</tr>
</tbody>
</table>

Remarks

By default, `value = SuperLU.ColumnOrderingMethod.ColumnApproximateMinimumDegree`. 
Exception

System.ArgumentException is thrown if value is not one of the above values.

ConditionNumber
public double ConditionNumber {get; }

Description
The estimate of the reciprocal condition number of the matrix $A$.

Property Value
A double scalar containing the reciprocal condition number of the matrix $A$ after equilibration and permutation of rows/columns (if done). If the reciprocal condition number is less than machine precision, in particular if it is equal to 0, the matrix is singular to working precision.

DiagonalPivotThreshold
public double DiagonalPivotThreshold {get; set; }

Description
The threshold used for a diagonal entry to be an acceptable pivot.

Property Value
A double scalar specifying the threshold used for a diagonal entry to be an acceptable pivot.

Remarks
By default, $\text{DiagonalPivotThreshold}=1.0$, i.e. classical partial pivoting.

Exception

System.ArgumentException is thrown if $\text{DiagonalPivotThreshold}$ is not in the interval $[0.0, 1.0]$.

Equilibrate
public bool Equilibrate {get; set; }

Description
Specifies if input matrix $A$ is equilibrated before factorization.

Property Value
A bool specifying whether or not matrix $A$ is equilibrated before factorization. If $\text{Equilibrate}$ is true the system is equilibrated, if $\text{Equilibrate}$ is false, no equilibration is performed.

Remarks
By default, $\text{Equilibrate} = \text{true}$.

EquilibrationMethod
public Imsl.Math.SuperLU.Scaling EquilibrationMethod {get; }

Description
The type of equilibration used before matrix factorization.
Property Value
A Scaling value specifying the equilibration option used.

Remarks

<table>
<thead>
<tr>
<th>EquilibrationMethod</th>
<th>Option description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No equilibration is performed.</td>
</tr>
<tr>
<td>Row</td>
<td>Equilibration is performed with row scaling.</td>
</tr>
<tr>
<td>Column</td>
<td>Equilibration is performed with column scaling.</td>
</tr>
<tr>
<td>RowAndColumn</td>
<td>Equilibration is performed with row and column scaling.</td>
</tr>
</tbody>
</table>

ForwardErrorBound
public double ForwardErrorBound {get; }  

Description
The estimated forward error bound for the solution vector.

Property Value
A double containing the estimated forward error bound for the solution vector. The estimate is as reliable as the estimate for the reciprocal condition number, and is almost always a slight overestimate of the true error. If iterative refinement is not used, the return value is set to 1.0.

IterativeRefinement
public bool IterativeRefinement {get; set; }  

Description
The iterative refinement option.

Property Value
A bool specifying whether to use iterative refinement.

Remarks
For iterative refinement, set IterativeRefinement = true, otherwise set IterativeRefinement = false.

By default, IterativeRefinement = false.

PivotGrowth
public bool PivotGrowth {get; set; }  

Description
Specifies whether to compute the reciprocal pivot growth factor.

Property Value
A bool specifying whether to compute the reciprocal pivot growth factor. true indicates the reciprocal pivot factor will be computed, false indicates it will not be computed.
Remarks
By default, PivotGrowth = false.

**ReciprocalPivotGrowthFactor**
`public double ReciprocalPivotGrowthFactor {get; }`

**Description**
The reciprocal pivot growth factor.

**Property Value**
A double scalar representing the reciprocal growth factor

\[
\max_{1 \leq j \leq n} \frac{\|\tilde{A}_j\|_\infty}{\|U_j\|_\infty}.
\]

If the value is much less than 1, the stability of the \(LU\) factorization could be poor.

**RelativeBackwardError**
`public double RelativeBackwardError {get; }`

**Description**
The componentwise relative backward error of the solution vector.

**Property Value**
A double containing the componentwise relative backward error of the solution vector \(x\). If IterativeRefinement is not set to true, then RelativeBackwardError returns 1.0.

**SymmetricMode**
`public bool SymmetricMode {get; set; }`

**Description**
The symmetric mode option.

**Property Value**
A bool indicating if symmetric mode is to be used.

Remarks
The symmetric mode option should be applied if the input matrix \(A\) is diagonally dominant or nearly so. The user should then define a small diagonal pivot threshold (e.g. 0.0 or 0.01) by property DiagonalPivotThreshold and choose an \((A^T + A)\)-based column permutation algorithm (e.g. column permutation method SuperLU.ColumnOrdering.MinimumDegreeAtPlusA). SymmetricMode=true implies symmetric mode is used.

By default, SymmetricMode=false.

**Constructor**

**SuperLU**
`public SuperLU(Imsl.Math.SparseMatrix A)`
Description
Constructor for SuperLU.

Parameter

\( A \) – A SparseMatrix containing the sparse square input matrix.

## Methods

### GetPerformanceTuningParameters

```
public int GetPerformanceTuningParameters(Imsl.Math.SuperLU.PerformanceParameters parameter)
```

**Description**
Returns a performance tuning parameter value.

**Parameter**

parameter – A PerformanceParameters scalar that specifies the parameter for which the value is to be returned.

<table>
<thead>
<tr>
<th>parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanelSize</td>
<td>The panel size.</td>
</tr>
<tr>
<td>RelaxationParameter</td>
<td>The relaxation parameter to control supernode amalgamation.</td>
</tr>
<tr>
<td>MaximumSupernodeSize</td>
<td>The maximum allowable size for a supernode.</td>
</tr>
<tr>
<td>MinimumRowDimension</td>
<td>The minimum row dimension to be used for 2D blocking.</td>
</tr>
<tr>
<td>MinimumColumnDimension</td>
<td>The minimum column dimension to be used for 2D blocking.</td>
</tr>
<tr>
<td>FillFactor</td>
<td>The estimated fill factor for ( L ) and ( U ), compared with ( A ).</td>
</tr>
</tbody>
</table>

**Returns**
An int containing the current value used for the specified tuning parameter.

### SetPerformanceTuningParameters

```
public void SetPerformanceTuningParameters(Imsl.Math.SuperLU.PerformanceParameters parameter, int tunedValue)
```

**Description**
Sets performance tuning parameters.
Parameters

parameter – A PerformanceParameters scalar that specifies the parameter to be tuned.
tunedValue – An int scalar that specifies the value to be used for the specified tuning parameter.

<table>
<thead>
<tr>
<th>parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanelSize</td>
<td>The panel size.</td>
<td>10</td>
</tr>
<tr>
<td>RelaxationParameter</td>
<td>The relaxation parameter to control supernode amalgamation.</td>
<td>5</td>
</tr>
<tr>
<td>MaximumSupernodeSize</td>
<td>The maximum allowable size for a supernode.</td>
<td>100</td>
</tr>
<tr>
<td>MinimumRowDimension</td>
<td>The minimum row dimension to be used for 2D blocking.</td>
<td>200</td>
</tr>
<tr>
<td>MinimumColumnDimension</td>
<td>The minimum column dimension to be used for 2D blocking.</td>
<td>40</td>
</tr>
<tr>
<td>FillFactor</td>
<td>The estimated fill factor for L and U, compared with A.</td>
<td>20</td>
</tr>
</tbody>
</table>

Exception

System.ArgumentException is thrown if tunedValue is not greater than zero.

Solve

public double[] Solve(double[] b)

Description
Computation of the solution vector for the system \(Ax = b\).

Parameter

b – A double vector of length \(n\), \(n\) the order of input matrix \(A\), containing the right hand side.

Returns
A double vector containing the solution to the system \(Ax = b\). Optionally, the solution is improved by iterative refinement, if IterativeRefinement = true. Method Solve internally first factorizes matrix \(A\) (step 1 of the introduction) if the factorization has not been done before.

SolveTranspose

public double[] SolveTranspose(double[] b)

Description
Computation of the solution vector for the system \(A^Tx = b\).

Parameter

b – A double vector of length \(n\) containing the right hand side, \(n\) is the order of input matrix \(A\).

Returns
A double vector containing the solution to the system \(A^Tx = b\). Optionally, the solution is improved by iterative refinement, if IterativeRefinement = true. Method SolveTranspose internally first factorizes matrix \(A\) (step 1 of the introduction) if the factorization has not been done before.
Example: LU Factorization of a Sparse Matrix

The LU Factorization of the sparse 6 × 6 matrix

\[
A = \begin{pmatrix}
10.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 10.0 & -3.0 & -1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 15.0 & 0.0 & 0.0 & 0.0 \\
-2.0 & 0.0 & 0.0 & 10.0 & -1.0 & 0.0 \\
-1.0 & 0.0 & 0.0 & -5.0 & 1.0 & -3.0 \\
-1.0 & -2.0 & 0.0 & 0.0 & 0.0 & 6.0
\end{pmatrix}
\]

is computed. The sparse coordinate form for A is given by a series of row, column, value triplets:

<table>
<thead>
<tr>
<th>row</th>
<th>column</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>10.0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>10.0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-3.0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>-1.0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>15.0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-2.0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>10.0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>-1.0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-1.0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>-5.0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-3.0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>-1.0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-2.0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6.0</td>
</tr>
</tbody>
</table>

Let

\[ y^T = (1.0, 2.0, 3.0, 4.0, 5.0, 6.0) \]

so that

\[ b_1 := Ay = (10.0, 7.0, 45.0, 33.0, -34.0, 31.0)^T \]

and

\[ b_2 := A^T y = (-9.0, 8.0, 39.0, 13.0, 1.0, 21.0)^T. \]

The LU factorization of A is used to solve the sparse linear systems \( Ax = b_1 \) and \( A^T x = b_2 \) with iterative refinement. The reciprocal pivot growth factor and the reciprocal condition number are also computed.
SuperLU slu;
double conditionNumber, recip_pivot_growth;
double[] sol = null;
double Ferr, Berr;

double[] b1 = {10.0, 7.0, 45.0, 33.0, -34.0, 31.0};
double[] b2 = {-9.0, 8.0, 39.0, 13.0, 1.0, 21.0};

// Initialize matrix A.
m = 6;
SparseMatrix a = new SparseMatrix(m, m);

a.Set(0, 0, 10.0);
a.Set(1, 1, 10.0);
a.Set(1, 2, -3.0);
a.Set(1, 3, -1.0);
a.Set(2, 2, 15.0);
a.Set(3, 0, -2.0);
a.Set(3, 3, 10.0);
a.Set(3, 4, -1.0);
a.Set(4, 0, -1.0);
a.Set(4, 3, -5.0);
a.Set(4, 4, 1.0);
a.Set(4, 5, -3.0);
a.Set(5, 0, -1.0);
a.Set(5, 1, -2.0);
a.Set(5, 5, 6.0);

// Compute the sparse LU factorization of a
slu = new SuperLU(a);
slu.Equilibrate = false;
slu.ColumnOrderingMethod = SuperLU.ColumnOrdering.Natural;
slu.PivotGrowth = true;

// Set option of iterative refinement
slu.IterativeRefinement = true;

// Solve sparse system A*x = b1
Console.Out.WriteLine();
Console.Out.WriteLine("Solve sparse System Ax=b1");
Console.Out.WriteLine("================================");
Console.Out.WriteLine();
sol = slu.Solve(b1);
new PrintMatrix("Solution").Print(sol);

Ferr = slu.ForwardErrorBound;
Berr = slu.RelativeBackwardError;

Console.Out.WriteLine();
Console.Out.WriteLine("Forward error bound: " + Ferr);
Console.Out.WriteLine("Relative backward error: " + Berr);
Console.Out.WriteLine();
Console.Out.WriteLine();
// Solve sparse system (A^T)*x = b2
Console.Out.WriteLine();
Console.Out.WriteLine("Solve sparse System (A^T)*x=b2");
Console.Out.WriteLine("==============================");
Console.Out.WriteLine();
sol = slu.SolveTranspose(b2);
new PrintMatrix("Solution").Print(sol);

Ferr = slu.ForwardErrorBound;
Berr = slu.RelativeBackwardError;

System.Console.Out.WriteLine();
System.Console.Out.WriteLine("Forward error bound: "+Ferr);
System.Console.Out.WriteLine("Relative backward error: "+Berr);
System.Console.Out.WriteLine();
System.Console.Out.WriteLine();

// Compute reciprocal pivot growth factor and condition number
recip_pivot_growth = slu.ReciprocalPivotGrowthFactor;
conditionNumber = slu.ConditionNumber;

System.Console.Out.WriteLine("Pivot growth factor and condition number");
System.Console.Out.WriteLine("========================================");
System.Console.Out.WriteLine();
  recip_pivot_growth);
System.Console.Out.WriteLine("Reciprocal condition number: " + conditionNumber);
System.Console.Out.WriteLine();
System.Console.Out.WriteLine();

Output
Solve sparse System Ax=b1
========================================
Solution
  0 1
  1 2
  2 3
  3 4
  4 5
  5 6

Forward error bound: 2.74489425897801E-14
Relative backward error: 0
Solve sparse System \((A^T)x = b\)

Solution

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
3 & 4 & 5 & 6 \\
\end{array}
\]

Forward error bound: 2.41379101136191E-15
Relative backward error: 0

Pivot growth factor and condition number

Reciprocal pivot growth factor: 1
Reciprocal condition number: 0.0244510978043912

---

**SuperLU.ColumnOrdering Enumeration**

public enumeration Imsl.Math.SuperLU.ColumnOrdering
The column permutation method to be used.

**Fields**

- **ColumnApproximateMinimumDegree**
  public Imsl.Math.SuperLU.ColumnOrdering ColumnApproximateMinimumDegree
  
  Description
  For column ordering, use column approximate minimum degree ordering.

- **MinimumDegreeAtA**
  public Imsl.Math.SuperLU.ColumnOrdering MinimumDegreeAtA
  
  Description
  For column ordering, use minimum degree ordering on the structure of \(A^T A\).
**MinimumDegreeAtPlusA**

public Imsl.Math.SuperLU.ColumnOrdering MinimumDegreeAtPlusA

**Description**
For column ordering, use minimum degree ordering on the structure of $A^T + A$.

**Natural**

public Imsl.Math.SuperLU.ColumnOrdering Natural

**Description**
For column ordering, use the natural ordering.

---

**SuperLU.PerformanceParameters Enumeration**

public enumeration Imsl.Math.SuperLU.PerformanceParameters

Performance tuning parameters which can be adjusted via method SetPerformanceTuningParameters.

**Fields**

**FillFactor**

public Imsl.Math.SuperLU.PerformanceParameters FillFactor

**Description**
The estimated fill factor for $L$ and $U$, compared with $A$.

**MaximumSupernodeSize**

public Imsl.Math.SuperLU.PerformanceParameters MaximumSupernodeSize

**Description**
The maximum allowable size for a supernode.

**MinimumColumnDimension**

public Imsl.Math.SuperLU.PerformanceParameters MinimumColumnDimension

**Description**
The minimum column dimension to be used for 2D blocking.

**MinimumRowDimension**

public Imsl.Math.SuperLU.PerformanceParameters MinimumRowDimension
Description
The minimum row dimension to be used for 2D blocking.

**PanelSize**

```csharp
public Imsl.Math.SuperLU.PerformanceParameters PanelSize
```

Description
The panel size, that is, the number of consecutive columns of \(A\) being factorized at a time.

**RelaxationParameter**

```csharp
public Imsl.Math.SuperLU.PerformanceParameters RelaxationParameter
```

Description
The relaxation parameter to control supernode amalgamation.

---

**SuperLU.Scaling Enumeration**

```csharp
public enumeration Imsl.Math.SuperLU.Scaling
```

One of several possible equilibration option return values for property EquilibrationMethod.

**Fields**

**Column**

```csharp
public Imsl.Math.SuperLU.Scaling Column
```

Description
Indicates that input matrix \(A\) was column scaled before factorization. This is a return value for EquilibrationMethod.

**None**

```csharp
public Imsl.Math.SuperLU.Scaling None
```

Description
Indicates that input matrix \(A\) was not equilibrated before factorization. This is a return value for EquilibrationMethod.

**Row**

```csharp
public Imsl.Math.SuperLU.Scaling Row
```
**Description**

Indicates that input matrix $A$ was row scaled before factorization. This is a return value for `EquilibrationMethod`.

**RowAndColumn**

```java
public Imsl.Math.SuperLU.Scaling RowAndColumn
```

**Description**

Indicates that input matrix $A$ was row and column scaled before factorization. This is a return value for `EquilibrationMethod`.

---

**ComplexSuperLU Class**

```java
public class Imsl.Math.ComplexSuperLU
```

Computes the LU factorization of a general sparse matrix of type `ComplexSparseMatrix` by a column method and solves a sparse linear system of equations $Ax = b$.

Consider the sparse linear system of equations

$$Ax = b.$$ 

Here, $A$ is a general square, nonsingular, $n$ by $n$ sparse matrix, and $x$ and $b$ are vectors of length $n$. All entries in $A$, $x$ and $b$ are of type `Complex`.

Gaussian elimination, applied to the system above, can be shortly described as follows:

1. Compute a triangular factorization $P_rD_rAD_cP_c = LU$. Here, $D_r$ and $D_c$ are positive definite diagonal matrices to equilibrate the system and $P_r$ and $P_c$ are permutation matrices to ensure numerical stability and preserve sparsity. $L$ is a unit lower triangular matrix and $U$ is an upper triangular matrix.

2. Solve $Ax = b$ by evaluating

$$x = A^{-1}b = D_c(P_r(U^{-1}(L^{-1}(P_c(D_r)b)))).$$

This is done efficiently by multiplying from right to left in the last expression: Scale the rows of $b$ by $D_r$. Multiplying $P_c(D_r)b$ means permuting the rows of $D_r b$. Multiplying $L^{-1}(P_c(D_r)b)$ means solving the triangular system of equations with matrix $L$ by substitution. Similarly, multiplying $U^{-1}(L^{-1}(P_c(D_r)b))$ means solving the triangular system with $U$.

Class `ComplexSuperLU` handles step 1 above in the `Solve` method if it has not been computed prior to step 2. More precisely, before $Ax = b$ is solved the following steps are performed:

1. Equilibrate matrix $A$, i.e. compute diagonal matrices $D_r$ and $D_c$ so that $\hat{A} = D_rAD_c$ is “better conditioned” than $A$, i.e. $\hat{A}^{-1}$ is less sensitive to perturbations in $\hat{A}$ than $A^{-1}$ is to perturbations in $A$. 

---

**Linear Systems**

ComplexSuperLU • 63
2. Order the columns of $\hat{A}$ to increase the sparsity of the computed $L$ and $U$ factors, i.e. replace $\hat{A}$ by $\hat{A}P_c$ where $P_c$ is a column permutation matrix.

3. Compute the $LU$ factorization of $\hat{A}P_c$. For numerical stability, the rows of $\hat{A}P_c$ are eventually permuted through the factorization process by scaled partial pivoting, leading to the decomposition $\tilde{A} := P_r\hat{A}P_c = LU$. The $LU$ factorization is done by a left looking supernode-panel algorithm with 2-D blocking. See Demmel, Eisenstat, Gilbert et al. (1999) for further information on this technique.

4. Compute the reciprocal pivot growth factor

$$\max_{1 \leq j \leq n} \frac{\|A_j\|_\infty}{\|U_j\|_\infty},$$

where $A_j$ and $U_j$ denote the $j$-th column of matrices $\tilde{A}$ and $U$, respectively.

5. Estimate the reciprocal of the condition number of matrix $\tilde{A}$.

Method Solve uses this information to perform the following steps:

1. Solve the system $Ax = b$ using the computed triangular factors.

2. Iteratively refine the solution, again using the computed triangular factors. This is equivalent to Newton’s method.

3. Compute forward and backward error bounds for the solution vector $x$.

Some of the steps mentioned above are optional. Their settings can be controlled by the Set methods and properties of class ComplexSuperLU.

Class ComplexSuperLU is based on the SuperLU code written by Demmel, Gilbert, Li et al. For more detailed explanations of the factorization and solve steps, see the SuperLU Users’ Guide (1999).

Copyright (c) 2003, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from U.S. Dept. of Energy)
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Properties

**ColumnOrderingMethod**

```csharp
public Imsl.Math.ComplexSuperLU.ColumnOrdering ColumnOrderingMethod {get; set; }
```

**Description**

The method used to permute the columns of the input matrix.

**Property Value**

A `ColumnOrdering` scalar specifying how the columns of the input matrix are to be permuted for sparsity preservation.

<table>
<thead>
<tr>
<th>value</th>
<th>method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural</td>
<td>natural ordering, that is $P_c = I$, $I$ the</td>
</tr>
<tr>
<td></td>
<td>identity matrix</td>
</tr>
<tr>
<td>MinimumDegreeAtPlusA</td>
<td>minimum degree ordering on the structure of $A^T + A$</td>
</tr>
<tr>
<td>MinimumDegreeAtA</td>
<td>minimum degree ordering on the structure of $A^T A$</td>
</tr>
<tr>
<td>ColumnApproximateMinimumDegree</td>
<td>column approximate minimum degree ordering</td>
</tr>
</tbody>
</table>

**Remarks**

By default, `ColumnOrderingMethod = ComplexSuperLU.ColumnOrdering.ColumnApproximateMinimumDegree`.

**Exception**

`System.ArgumentException` is thrown if `value` is not one of the above values.

**ConditionNumber**

```csharp
public double ConditionNumber {get; }
```

**Description**

The estimate of the reciprocal condition number of the matrix $A$. 

**Linear Systems**  ComplexSuperLU • 65
**Property Value**
A double scalar containing the reciprocal condition number of the matrix A after equilibration and permutation of rows/columns (if done). If the reciprocal condition number is less than machine precision, in particular if it is equal to 0, the matrix is singular to working precision.

**DiagonalPivotThreshold**
public double DiagonalPivotThreshold {get; set; }

**Description**
The threshold used for a diagonal entry to be an acceptable pivot.

**Property Value**
A double scalar specifying the threshold used for a diagonal entry to be an acceptable pivot.

**Remarks**
By default, DiagonalPivotThreshold=1.0, i.e. classical partial pivoting.

**Exception**
System.ArgumentException is thrown if DiagonalPivotThreshold is not in the interval [0.0, 1.0].

**Equilibrate**
public bool Equilibrate {get; set; }

**Description**
Specifies if input matrix A is equilibrated before factorization.

**Property Value**
A bool specifying whether or not matrix A is equilibrated before factorization. If Equilibrate is true the system is equilibrated, if Equilibrate is false, no equilibration is performed.

**Remarks**
By default, Equilibrate = true.

**EquilibrationMethod**
public Imsl.Math.ComplexSuperLU.Scaling EquilibrationMethod {get; }

**Description**
The type of equilibration used before matrix factorization.

**Property Value**
A Scaling value specifying the equilibration option used.

**Remarks**

<table>
<thead>
<tr>
<th>EquilibrationMethod</th>
<th>Option description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No equilibration is performed.</td>
</tr>
<tr>
<td>Row</td>
<td>Equilibration is performed with row scaling.</td>
</tr>
<tr>
<td>Column</td>
<td>Equilibration is performed with column scaling.</td>
</tr>
<tr>
<td>RowAndColumn</td>
<td>Equilibration is performed with row and column scaling.</td>
</tr>
</tbody>
</table>
ForwardErrorBound
public double ForwardErrorBound {get; }

Description
The estimated forward error bound for the solution vector.

Property Value
A double containing the estimated forward error bound for the solution vector. The estimate is as reliable as the estimate for the reciprocal condition number, and is almost always a slight overestimate of the true error. If iterative refinement is not used, the return value is set to 1.0.

IterativeRefinement
public bool IterativeRefinement {get; set; }

Description
Specifies whether to perform iterative refinement.

Property Value
A bool specifying whether to use iterative refinement.

Remarks
For iterative refinement, set IterativeRefinement = true, otherwise set IterativeRefinement = false.
By default, IterativeRefinement = false.

PivotGrowth
public bool PivotGrowth {get; set; }

Description
Specifies whether to compute the reciprocal pivot growth factor.

Property Value
A bool specifying whether to compute the reciprocal pivot growth factor. true indicates the reciprocal pivot factor will be computed, false indicates it will not be computed.

Remarks
By default, PivotGrowth = false.

ReciprocalPivotGrowthFactor
public double ReciprocalPivotGrowthFactor {get; }

Description
The reciprocal pivot growth factor.
**Property Value**

A double scalar representing the reciprocal growth factor

\[
\max_{1 \leq j \leq n} \frac{\| \tilde{A}_j \|_\infty}{\| U_j \|_\infty}.
\]

If the value is much less than 1, the stability of the LU factorization could be poor.

**RelativeBackwardError**

```csharp
public double RelativeBackwardError {get; }
```

**Description**

The componentwise relative backward error of the solution vector.

**Property Value**

A double containing the componentwise relative backward error of the solution vector \(x\). If \(IterativeRefinement\) is not set to true, then \(RelativeBackwardError\) returns 1.0.

**SymmetricMode**

```csharp
public bool SymmetricMode {get; set; }
```

**Description**

Specifies whether to use the symmetric mode.

**Property Value**

A bool indicating if symmetric mode is to be used.

**Remarks**

The symmetric mode option should be applied if the input matrix \(A\) is diagonally dominant or nearly so. The user should then define a small diagonal pivot threshold (e.g. 0.0 or 0.01) by property \(DiagonalPivotThreshold\) and choose an \((A^T + A)\)-based column permutation algorithm (e.g. column permutation method \(SuperLU.ColumnOrdering.MinimumDegreeAtA\)). \(SymmetricMode=true\) implies symmetric mode is used.

By default, \(SymmetricMode=false\).

**Constructor**

```csharp
public ComplexSuperLU(Imsl.Math.ComplexSparseMatrix A)
```

**Description**

Constructor for \(ComplexSuperLU\).

**Parameter**

\(A\) – A \(ComplexSparseMatrix\) containing the sparse square input matrix.
Methods

GetPerformanceTuningParameters

public int GetPerformanceTuningParameters(Imsl.Math.ComplexSuperLU.PerformanceParameters parameter)

Description

Returns a performance tuning parameter value.

Parameter

parameter – a PerformanceParameters scalar that specifies the parameter for which the value is to be returned.

<table>
<thead>
<tr>
<th>parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanelSize</td>
<td>The panel size.</td>
</tr>
<tr>
<td>RelaxationParameter</td>
<td>The relaxation parameter to control supernode amalgamation.</td>
</tr>
<tr>
<td>MaximumSupernodeSize</td>
<td>The maximum allowable size for a supernode.</td>
</tr>
<tr>
<td>MinimumRowDimension</td>
<td>The minimum row dimension to be used for 2D blocking.</td>
</tr>
<tr>
<td>MinimumColumnDimension</td>
<td>The minimum column dimension to be used for 2D blocking.</td>
</tr>
<tr>
<td>FillFactor</td>
<td>The estimated fill factor for L and U, compared with A.</td>
</tr>
</tbody>
</table>

Returns

An int containing the current value used for the specified tuning parameter.

SetPerformanceTuningParameters

public void SetPerformanceTuningParameters(Imsl.Math.ComplexSuperLU.PerformanceParameters parameter, int tunedValue)

Description

Sets performance tuning parameters.

Parameters

parameter – A PerformanceParameters that specifies the parameter to be tuned.

tunedValue – An int scalar that specifies the value to be used for the specified tuning parameter.
<table>
<thead>
<tr>
<th>parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanelSize</td>
<td>The panel size.</td>
<td>10</td>
</tr>
<tr>
<td>RelaxationParameter</td>
<td>The relaxation parameter to control supernode amalgamation.</td>
<td>5</td>
</tr>
<tr>
<td>MaximumSupernodeSize</td>
<td>The maximum allowable size for a supernode.</td>
<td>100</td>
</tr>
<tr>
<td>MinimumRowDimension</td>
<td>The minimum row dimension to be used for 2D blocking.</td>
<td>200</td>
</tr>
<tr>
<td>MinimumColumnDimension</td>
<td>The minimum column dimension to be used for 2D blocking.</td>
<td>40</td>
</tr>
<tr>
<td>FillFactor</td>
<td>The estimated fill factor for $L$ and $U$, compared with $A$.</td>
<td>20</td>
</tr>
</tbody>
</table>

**Exception**

`System.ArgumentException` is thrown if `tunedValue` is not greater than zero.

**Solve**

```csharp
```

**Description**

Computation of the solution vector for the system $Ax = b$.

**Parameter**

$b$ – A `Complex` vector of length $n$, $n$ the order of input matrix $A$, containing the right hand side.

**Returns**

A `Complex` vector containing the solution to the system $Ax = b$. Optionally, the solution is improved by iterative refinement, if `IterativeRefinement` is set to `true`. Method `Solve` internally first factorizes matrix $A$ (step 1 of the introduction) if the factorization has not been done before.

**SolveConjugateTranspose**

```csharp
```

**Description**

Computation of the solution vector for the system $A^H x = b$.

**Parameter**

$b$ – A `Complex` vector of length $n$, $n$ the order of input matrix $A$, containing the right hand side.

**Returns**

A `Complex` vector containing the solution to the system $A^H x = b$. Optionally, the solution is improved by iterative refinement, if `IterativeRefinement` is set to `true`. Method `SolveConjugateTranspose` internally first factorizes matrix $A$ (step 1 of the introduction) if the factorization has not been done before.

**SolveTranspose**

```csharp
```
Description
Computation of the solution vector for the system $A^T x = b$.

Parameter
b – A Complex vector of length $n$ containing the right hand side, $n$ is the order of input matrix $A$.

Returns
A Complex vector containing the solution to the system $A^T x = b$. Optionally, the solution is improved by iterative refinement, if IterativeRefinement is set to true. Method SolveTranspose internally first factorizes matrix $A$ (step 1 of the introduction) if the factorization has not been done before.

Example: LU Factorization of a Complex Sparse Matrix
The LU Factorization of the sparse complex $6 \times 6$ matrix

$$A = \begin{pmatrix}
10 + 7i & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 3 + 2i & -3 + 0i & -1 + 2i & 0.0 & 0.0 \\
0.0 & 0.0 & 4 + 2i & 0.0 & 0.0 & 0.0 \\
-2 - 4i & 0.0 & 0.0 & 1 + 6i & -1 + 3i & 0.0 \\
-5 + 4i & 0.0 & 0.0 & -5 + 0i & 12 + 2i & -7 + 7i \\
-1 + 12i & -2 + 8i & 0.0 & 0.0 & 0.0 & 3 + 7i \\
\end{pmatrix}$$

is computed. The sparse coordinate form for $A$ is given by row, column, value triplets:

<table>
<thead>
<tr>
<th>row</th>
<th>column</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>10+7i</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3+2i</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-3+0i</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>-1+2i</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4+2i</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-2-4i</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1+6i</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>-1+3i</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-5+4i</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>-5+0i</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>12+2i</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-7+7i</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>-1+12i</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-2+8i</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3+7i</td>
</tr>
</tbody>
</table>

Let

$$x^T = (1 + i, 2 + 2i, 3 + 3i, 4 + 4i, 5 + 5i, 6 + 6i)$$

so that

$$b_1 := Ax = (3 + 17i, -19 + 5i, 6 + 18i, -38 + 32i, -63 + 49i, -57 + 83i)^T$$

and

$$b_2 := A^H x = (54 - 112i, 46 - 58i, 12, 5 - 51i, 78 + 34i, 60 - 94i)^T.$$
The LU factorization of $A$ is used to solve the complex sparse linear systems $Ax = b_1$ and $A^Hx = b_2$ with iterative refinement. The reciprocal pivot growth factor and the reciprocal condition number are also computed.

```csharp
using System;
using Imsl.Math;

public class ComplexSuperLUEx1
{
    public static void Main(String[] args)
    {
        int m;
        ComplexSuperLU cslu;
        double conditionNumber, recip_pivot_growth;
        Complex[] sol = null;
        double Ferr, Berr;
        Complex[] b1 = { new Complex(3.0, 17.0), new Complex(-19.0, 5.0),
                         new Complex(6.0, 18.0), new Complex(-38.0, 32.0),
                         new Complex(-63.0, 49.0), new Complex(-57.0, 83.0)
                     };
        Complex[] b2 = { new Complex(54.0, -112.0), new Complex(46.0, -58.0),
                         new Complex(12.0, 0.0), new Complex(5.0, -51.0),
                         new Complex(78.0, 34.0), new Complex(60.0, -94.0)
                     };
        // Initialize input matrix A.
        m = 6;
        ComplexSparseMatrix a = new ComplexSparseMatrix(m, m);
        a.Set(0, 0, new Complex(10.0, 7.0));
        a.Set(1, 1, new Complex(3.0, 2.0));
        a.Set(1, 2, new Complex(-3.0, 0.0));
        a.Set(1, 3, new Complex(-1.0, 2.0));
        a.Set(2, 2, new Complex(4.0, 2.0));
        a.Set(3, 0, new Complex(-2.0, -4.0));
        a.Set(3, 3, new Complex(1.0, 6.0));
        a.Set(3, 4, new Complex(-1.0, 3.0));
        a.Set(4, 0, new Complex(-5.0, 4.0));
        a.Set(4, 3, new Complex(-5.0, 0.0));
        a.Set(4, 4, new Complex(12.0, 2.0));
        a.Set(4, 5, new Complex(-7.0, 7.0));
        a.Set(5, 0, new Complex(-1.0, 12.0));
        a.Set(5, 1, new Complex(-2.0, 8.0));
        a.Set(5, 5, new Complex(3.0, 7.0));
        // Compute the sparse LU factorization of a
        cslu = new ComplexSuperLU(a);
        cslu.Equilibrate = false;
        cslu.ColumnOrderingMethod = ComplexSuperLU.ColumnOrdering.Natural;
        cslu.PivotGrowth = true;
        // Set option of iterative refinement
        cslu.IterativeRefinement = true;
        // Solve sparse system A*x = b1
        Console.Out.WriteLine();
    }
}
```
Console.Out.WriteLine("Solve sparse System A*x=b1");
Console.Out.WriteLine("================================");
Console.Out.WriteLine();
sol = cslu.Solve(b1);
new PrintMatrix("Solution").Print(sol);

// Determine error bounds
Ferr = cslu.ForwardErrorBound;
Berr = cslu.RelativeBackwardError;
Console.Out.WriteLine();
Console.Out.WriteLine("Forward error bound: " + Ferr);
Console.Out.WriteLine("Relative backward error: " + Berr);
Console.Out.WriteLine();

// Solve sparse system (A^H)*x = b2
Console.Out.WriteLine();
Console.Out.WriteLine("Solve sparse System (A^H)*x=b2");
Console.Out.WriteLine("================================");
Console.Out.WriteLine();
sol = cslu.SolveConjugateTranspose(b2);
new PrintMatrix("Solution").Print(sol);

// Determine error bounds
Ferr = cslu.ForwardErrorBound;
Berr = cslu.RelativeBackwardError;
Console.Out.WriteLine();
Console.Out.WriteLine("Forward error bound: " + Ferr);
Console.Out.WriteLine("Relative backward error: " + Berr);
Console.Out.WriteLine();

// Compute reciprocal pivot growth factor and condition number
recip_pivot_growth = cslu.ReciprocalPivotGrowthFactor;
conditionNumber = cslu.ConditionNumber;
Console.Out.WriteLine("Pivot growth factor and condition number");
Console.Out.WriteLine("========================================");
Console.Out.WriteLine("Reciprocal pivot growth factor: " + recip_pivot_growth);
Console.Out.WriteLine("Reciprocal condition number: " + conditionNumber);
Console.Out.WriteLine();

Output

Solve sparse System A*x=b1
================================

Solution
0
0 1+i
1 2+2i
2 3+3i
3 4+4i
Solve sparse System \((A^H)x=b\)

==============================

Solution

0

0 1+i
1 2+2.00000000000001i
2 3+3i
3 4+4i
4 5+5i
5 6+6i

Forward error bound: 8.54834098797111E-15
Relative backward error: 1.02977208081174E-16

Pivot growth factor and condition number

Reciprocal pivot growth factor: 0.799382716049383
Reciprocal condition number: 0.0700654479096751

---

**ComplexSuperLU.ColumnOrdering**

**Enumeration**

public enumeration Imsl.Math.ComplexSuperLU.ColumnOrdering

The column permutation method to be used.

**Fields**

**ColumnApproximateMinimumDegree**

public Imsl.Math.ComplexSuperLU.ColumnOrdering ColumnApproximateMinimumDegree

**Description**

Indicates column approximate minimum degree ordering.
MinimumDegreeAtA
public Imsl.Math.ComplexSuperLU.ColumnOrdering MinimumDegreeAtA

Description
Indicates minimum degree ordering on the structure of $A^T A$.

MinimumDegreeAtPlusA
public Imsl.Math.ComplexSuperLU.ColumnOrdering MinimumDegreeAtPlusA

Description
Indicates minimum degree ordering on the structure of $A^T + A$.

Natural
public Imsl.Math.ComplexSuperLU.ColumnOrdering Natural

Description
Indicates natural ordering.

ComplexSuperLU.PerformanceParameters
Enumeration

public enumeration Imsl.Math.ComplexSuperLU.PerformanceParameters

Performance tuning parameters which can be adjusted via method
SetPerformanceTuningParameters.

Fields

FillFactor
public Imsl.Math.ComplexSuperLU.PerformanceParameters FillFactor

Description
The estimated fill factor for $L$ and $U$, compared with $A$.

MaximumSupernodeSize
public Imsl.Math.ComplexSuperLU.PerformanceParameters MaximumSupernodeSize
Description
The maximum allowable size for a supernode.

**MinimumColumnDimension**
public Imsl.Math.ComplexSuperLU.PerformanceParameters MinimumColumnDimension

Description
The minimum column dimension to be used for 2D blocking.

**MinimumRowDimension**
public Imsl.Math.ComplexSuperLU.PerformanceParameters MinimumRowDimension

Description
The minimum row dimension to be used for 2D blocking.

**PanelSize**
public Imsl.Math.ComplexSuperLU.PerformanceParameters PanelSize

Description
The panel size.

**RelaxationParameter**
public Imsl.Math.ComplexSuperLU.PerformanceParameters RelaxationParameter

Description
The relaxation parameter to control supernode amalgamation.

---

**ComplexSuperLU.Scaling Enumeration**

public enumeration Imsl.Math.ComplexSuperLU.Scaling

Equilibration method before factorization, this setting is returned from GetEquilibrationMethod.

**Fields**

**Column**
public Imsl.Math.ComplexSuperLU.Scaling Column
Description
Indicates that input matrix \( A \) was column scaled before factorization.

None
public Imsl.Math.ComplexSuperLU.Scaling None

Description
Indicates that input matrix \( A \) was not equilibrated before factorization.

Row
public Imsl.Math.ComplexSuperLU.Scaling Row

Description
Indicates that input matrix \( A \) was row scaled before factorization.

RowAndColumn
public Imsl.Math.ComplexSuperLU.Scaling RowAndColumn

Description
Indicates that input matrix \( A \) was row and column scaled before factorization.

Cholesky Class

public class Imsl.Math.Cholesky

Cholesky factorization of a matrix of type double.

Class Cholesky is based on the LINPACK routine SCHDC; see Dongarra et al. (1979).

Before the decomposition is computed, initial elements are moved to the leading part of \( A \) and final elements to the trailing part of \( A \). During the decomposition only rows and columns corresponding to the free elements are moved. The result of the decomposition is an lower triangular matrix \( R \) and a permutation matrix \( P \) that satisfy \( P^T A P = R R^T \), where \( P \) is represented by \( \text{ipvt} \).

The method Update is based on the LINPACK routine SCHUD; see Dongarra et al. (1979).

The Cholesky factorization of a matrix is \( A = RR^T \), where \( R \) is an lower triangular matrix. Given this factorization, Down date computes the factorization

\[
A - xx^T = \tilde{R}^T \tilde{R}
\]

Down date determines an orthogonal matrix \( U \) as the product \( G_N \ldots G_1 \) of Givens rotations, such that

\[
U \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ x^T \end{bmatrix}
\]
By multiplying this equation by its transpose and noting that $U^TU = I$, the desired result

$$R^T R - xx^T = R^T \tilde{R}$$

is obtained.

Let $a$ be the solution of the linear system $R^T a = x$ and let

$$\alpha = \sqrt{1 - \|a\|^2}$$

The Givens rotations, $G_i$, are chosen such that

$$G_1 \cdots G_N \begin{bmatrix} a \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The $G_i$, are $(N+1) \times (N+1)$ matrices of the form

$$G_i = \begin{bmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & c_i & 0 & -s_i \\ 0 & 0 & I_{N-i} & 0 \\ 0 & s_i & 0 & c_i \end{bmatrix}$$

where $I_k$ is the identity matrix of order $k$; and $c_i = \cos \theta_i, s_i = \sin \theta_i$ for some $\theta_i$.

The Givens rotations are then used to form

$$\tilde{R}, G_1 \cdots G_N \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ x^T \end{bmatrix}$$

The matrix

$$\tilde{R}$$

is lower triangular and

$$\tilde{x} = x$$

because

$$x = (R^T 0) \begin{bmatrix} a \\ \alpha \end{bmatrix} = (R^T 0) U^T U \begin{bmatrix} a \\ \alpha \end{bmatrix} = (R^T \tilde{x}) \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \tilde{x}.$$
**Description**
Create the Cholesky factorization of a symmetric positive definite matrix of type `double`.

**Parameter**
ap – A `double` square matrix to be factored.

**Exceptions**
- `Imsl.Math.SingularMatrixException` is thrown when the input matrix `a` is singular
- `Imsl.Math.NotSPDException` is thrown when the input matrix is not symmetric, positive definite.

**Methods**

**Downdate**
public void Downdate(double[] x)

**Description**
Downdates the factorization by subtracting a rank-1 matrix.

**Parameter**
x – A `double` array which specifies the rank-1 matrix. `x` is not modified by this function.

**Remarks**
The object will contain the Cholesky factorization of `a - x * transpose(x)`, where `a` is the previously factor matrix.

**Exception**
- `Imsl.Math.NotSPDException` is thrown if `a - x * transpose(x)` is not symmetric positive-definite.

**GetR**
public double[,] GetR()

**Description**
The R matrix that results from the Cholesky factorization.

**Returns**
A `double` matrix which contains the lower triangular R matrix that results from the Cholesky factorization, $A = RR^T$.

**Inverse**
public double[,] Inverse()

**Description**
Returns the inverse of this matrix.
Returns
A double matrix containing the inverse.

Solve
public double[] Solve(double[] b)

Description
Solve Ax = b where A is a positive definite matrix with elements of type double.

Parameter
   b – A double array containing the right-hand side of the linear system.

Returns
A double array containing the solution to the system of linear equations.

Update
public void Update(double[] x)

Description
Updates the factorization by adding a rank-1 matrix.

Parameter
   x – A double array which specifies the rank-1 matrix. x is not modified by this function.

Remarks
The object will contain the Cholesky factorization of a + x * transpose(x), where a is the previously factored matrix.

Example: Cholesky Factorization
The Cholesky Factorization of a matrix is performed as well as its inverse.

```csharp
using System;
using Imsl.Math;

public class CholeskyEx1
{
    public static void Main(String[] args)
    {
        double[,] a = {
            {1, - 3, 2},
            {- 3, 10, - 5},
            {2, - 5, 6}
        };
        double[] b = new double[]{27, - 78, 64};

        // Compute the Cholesky factorization of A
        Cholesky cholesky = new Cholesky(a);

        // Solve Ax = b
    }
}
```
double[] x = cholesky.Solve(b);
new PrintMatrix("x").Print(x);

// Find the inverse of A.
double[,] ainv = cholesky.Inverse();
new PrintMatrix("ainv").Print(ainv);
}

Output

x
0
0 1
1 -4
2 7

ainv
0 1 2
0 35 8 -5
1 8 2 -1
2 -5 -1 1

SparseCholesky Class

public class Imsl.Math.SparseCholesky
Sparse Cholesky factorization of a matrix of type SparseMatrix.

Class SparseCholesky computes the Cholesky factorization of a sparse symmetric positive definite matrix A. This factorization can then be used to compute the solution of the linear system \(Ax = b\).

Typically, the solution of a large sparse positive definite system \(Ax = b\) is done in 4 steps:

1. In step one, an ordering algorithm is used to preserve sparsity in the Cholesky factor \(L\) of matrix \(A\) during the numerical factorization process. The new order can be described by a permutation matrix \(P\).

2. Step two consists of setting up the data structure for the Cholesky factor \(L\), where \(PAP^T = LL^T\). This step is called the symbolic factorization phase of the computation. During symbolic factorization, only the sparsity pattern of sparse matrix \(A\), i.e., the locations of the nonzero entries of matrix \(A\) are needed but not any of the elements themselves.

3. In step 3, the numerical factorization phase, the Cholesky factorization is done numerically.

4. Step 4 is the solution phase. Here, the numerical solution, \(x\), to the original system is obtained by solving the two triangular systems \(Ly_1 = Pb\), \(L^T y_2 = y_1\) and the permutation \(x = P^T y_2\).
Class \texttt{SparseCholesky} realizes all four steps by algorithms described in George and Liu (1981). Especially, step one, is a realization of a minimum degree ordering algorithm. The numerical factorization in its standard form is based on a sparse compressed storage scheme. Alternatively, a multifrontal method can be used. The multifrontal method requires more storage but will be faster than the standard method in certain cases. The multifrontal method is based on the routines in Liu (1987). For a detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987) et al. (1987), and Liu (1986, 1989, 1992). The numerical factorization method can be specified by using the \texttt{NumericFactorizationMethod} (p. 82) property.

The \texttt{Solve} (p. 85) method will compute the symbolic and numeric factorizations if they have not already been computed or supplied by the user through \texttt{FactorSymbolically} (p. 84), \texttt{FactorNumerically} (p. 83), \texttt{SetNumericFactor} (p. 85), or \texttt{SetSymbolicFactor} (p. 85). These factorizations are retained for later use by the \texttt{Solve} method when different right-hand sides are to be solved.

There is a special situation where computations can be simplified. If an application generates different sparse symmetric positive definite coefficient matrices that all have the same sparsity pattern, then by using \texttt{SetSymbolicFactor} (p. 85) the symbolic factorization need only be computed once.

\section*{Properties}

\begin{description}
\item[\texttt{LargestDiagonalElement}] public double LargestDiagonalElement \{get; \}
\end{description}

\textbf{Description}

The largest diagonal element of the Cholesky factor.

\textbf{Property Value}

A \texttt{double} value specifying the largest diagonal element of the Cholesky factor. Use of this method is only sensible if a numeric factorization of the input matrix was done beforehand.

\begin{description}
\item[\texttt{NumberOfNonzeros}] public long NumberOfNonzeros \{get; \}
\end{description}

\textbf{Description}

The number of nonzeros in the Cholesky factor.

\textbf{Property Value}

A \texttt{long} containing the number of nonzeros (including the diagonal) of the Cholesky factor.

\begin{description}
\item[\texttt{NumericFactorizationMethod}] public Imsl.Math.SparseCholesky.NumericFactorization NumericFactorizationMethod \{get; set; \}
\end{description}

\textbf{Description}

The method used in the numerical factorization of the permuted input matrix.
Property Value

A NumericFactorization value equal to StandardMethod (p. 87) or MultiFrontalMethod (p. 87) representing the method used in the numeric factorization of the permuted input matrix.

Remarks

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StandardMethod</td>
<td>standard method as described by George/Liu (1981). This is the default.</td>
</tr>
<tr>
<td>MultiFrontalMethod</td>
<td>multifrontal method</td>
</tr>
</tbody>
</table>

SmallestDiagonalElement

public double SmallestDiagonalElement {get; }

Description

The smallest diagonal element of the Cholesky factor.

Property Value

A double value specifying the smallest diagonal element of the Cholesky factor. Use of this method is only sensible if a numeric factorization of the input matrix was done beforehand.

Constructor

SparseCholesky

public SparseCholesky(Imsl.Math.SparseMatrix A)

Description

Constructs the matrix structure for the Cholesky factorization of a sparse symmetric positive definite matrix of type SparseMatrix.

Parameter

A – The SparseMatrix symmetric positive definite matrix to be factored. Only the lower triangular part of the input matrix is used.

Methods

FactorNumerically

public void FactorNumerically()

Description

Computes the numeric factorization of a sparse real symmetric positive definite matrix.
Remarks
This method numerically factors the instance of the constructed matrix $A$, where $A$ is of type SparseMatrix and is symmetric positive definite. The factorization is obtained in several steps:

1. First, matrix $A$ is permuted to reduce fill-in, leading to a sparse symmetric positive definite matrix $PAP^T$.
2. Then, matrix $PAP^T$ is symbolically and numerically factored.

Note that the symbolic factorization is not done if the symbolic factor has been supplied by the user through the SetSymbolicFactor method.

Exception
Imsl.Math.NotSPDEception is thrown when the input matrix is not symmetric, positive definite.

FactorSymbolically
public void FactorSymbolically()

Description
Computes the symbolic factorization of a sparse real symmetric positive definite matrix.

Remarks
This method symbolically factors the instance of the constructed matrix $A$, where $A$ is of type SparseMatrix and is symmetric positive definite. The factorization is obtained in several steps:

1. First, matrix $A$ is permuted to reduce fill-in, leading to a sparse symmetric positive definite matrix $PAP^T$.
2. Then, matrix $PAP^T$ is symbolically factored.

Exception
Imsl.Math.NotSPDEception is thrown when the input matrix is not symmetric, positive definite.

GetNumericFactor

Description
Returns the numeric Cholesky factor.

Returns
A NumericFactor containing the numeric Cholesky factor.

GetSymbolicFactor
public Imsl.Math.SparseCholesky.SymbolicFactor GetSymbolicFactor()

Description
Returns the symbolic Cholesky factor.
**Returns**
A SymbolicFactor containing the symbolic Cholesky factor.

**SetNumericFactor**

**Description**
Sets the numeric Cholesky factor to use in solving of a sparse positive definite system of linear equations $Ax = b$.

**Parameter**
numericFactor – A NumericFactor containing the numeric Cholesky factor. By default the numeric factorization is computed.

**SetSymbolicFactor**
public void SetSymbolicFactor(Imsl.Math.SparseCholesky.SymbolicFactor symbolicFactor)

**Description**
Sets the symbolic Cholesky factor to use in solving a sparse positive definite system of linear equations $Ax = b$.

**Parameter**
symbolicFactor – A SymbolicFactor containing the symbolic Cholesky factor. By default the symbolic factorization is computed.

**Solve**
public double[] Solve(double[] b)

**Description**
Computes the solution of a sparse real symmetric positive definite system of linear equations $Ax = b$.

**Parameter**
b – A double vector of length equal to the order of matrix $A$ representing the right-hand side of the linear system.

**Returns**
A double vector of length equal to the order of matrix $A$ representing the solution to the system of linear equations $Ax = b$.

**Remarks**
This method solves the linear system $Ax = b$, where $A$ is symmetric positive definite. The solution is obtained in several steps:

1. First, matrix $A$ is permuted to reduce fill-in, leading to a sparse symmetric positive definite system $PAP^T(Px) = Pb$.
2. Then, matrix $PAP^T$ is symbolically and numerically factored.
3. The final solution is obtained by solving the systems \(Ly_1 = Pb, LT^Ty_2 = y_1\) and \(x = P^Ty_2\).

By default this method implements all of the above steps. The factorizations are retained for later use by subsequent solves. By choosing appropriate methods within this class, the computation can be reduced to the solution of the system \(Ax = b\) for a given or precomputed symbolic or numeric factor.

**Exception**

`Imsl.Math.NotSPDException` is thrown when the input matrix is not symmetric, positive definite.

**Example: Sparse Cholesky Factorization**

The Cholesky Factorization of a sparse symmetric positive definite matrix is computed. Some additional information about the Cholesky factorization is also computed.

```csharp
using System;
using Imsl.Math;

class SparseCholeskyEx1
{
    static void Main(String[] args)
    {
        SparseMatrix A = new SparseMatrix(5, 5);
        A.Set(0, 0, 10.0);
        A.Set(1, 1, 20.0);
        A.Set(2, 0, 1.0);
        A.Set(2, 2, 30.0);
        A.Set(3, 2, 4.0);
        A.Set(3, 3, 40.0);
        A.Set(4, 0, 2.0);
        A.Set(4, 1, 3.0);
        A.Set(4, 3, 5.0);
        A.Set(4, 4, 50.0);

        double[] b = { 55.0, 83.0, 103.0, 97.0, 82.0};
        SparseCholesky cholesky = new SparseCholesky(A);

        // Choose Multifrontal method as numeric factorization method

        // Compute solution
        double[] solution = cholesky.Solve(b);
        PrintMatrix("Computed Solution").Print(solution);

        // Print additional information about the factorization
        Console.Out.Write("Smallest diagonal element of Cholesky factor: ");
        Console.Out.WriteLine(cholesky.SmallestDiagonalElement);
        Console.Out.Write("Largest diagonal element of Cholesky factor: ");
        Console.Out.WriteLine(cholesky.LargestDiagonalElement);
        Console.Out.Write("Number of nonzeros in Cholesky factor: ");
        Console.Out.WriteLine(cholesky.NumberOfNonzeros);
    }
}
```
Output

Computed Solution
0
0 5
1 4
2 3
3 2
4 1

Smallest diagonal element of Cholesky factor: 3.16227766016838
Largest diagonal element of Cholesky factor: 7.01070609853244
Number of nonzeros in Cholesky factor: 11

SparseCholesky.NumericFactorization

Enumeration

public enumeration Imsl.Math.SparseCholesky.NumericFactorization

Numeric factorization methods.

Fields

MultiFrontalMethod

public Imsl.Math.SparseCholesky.NumericFactorization MultiFrontalMethod

Description
Indicates the multifrontal method will be used for numeric factorization.

StandardMethod


Description
Indicates the method of George/Liu (1981) will be used for numeric factorization.
**SparseCholesky.NumericFactor Class**

```
public class Imsl.Math.SparseCholesky.NumericFactor
The numeric Cholesky factorization of a matrix.
Used by GetNumericFactor and SetNumericFactor to hold the numeric Cholesky factorization of a matrix.
```

**SparseCholesky.SymbolicFactor Class**

```
public class Imsl.Math.SparseCholesky.SymbolicFactor
The symbolic Cholesky factorization of a matrix.
Used by GetSymbolicFactor and SetSymbolicFactor to hold the symbolic Cholesky factorization of a matrix.
```

**ComplexSparseCholesky Class**

```
public class Imsl.Math.ComplexSparseCholesky
Sparse Cholesky factorization of a matrix of type ComplexSparseMatrix.
Class ComplexSparseCholesky computes the Cholesky factorization of a sparse Hermitian positive definite matrix A. This factorization can then be used to compute the solution of the linear system Ax = b.
Typically, the solution of a large sparse positive definite system Ax = b is done in four steps.

1. In step one, an ordering algorithm is used to preserve sparsity in the Cholesky factor L of matrix A during the numerical factorization process. The new order can be described by a permutation matrix P.

2. Step two consists of setting up the data structure for the Cholesky factor L, where PAP^T = LL^T. This step is called the symbolic factorization phase of the computation. During symbolic factorization, only the sparsity pattern of sparse matrix A, i.e., the locations of the nonzero entries of matrix A are needed but not any of the elements themselves.

3. In step 3, the numerical factorization phase, the Cholesky factorization is done numerically.
```
4. Step 4 is the solution phase. Here, the numerical solution, $x$, to the original system is obtained by solving the two triangular systems $L y_1 = Pb, LT y_2 = y_1$ and the permutation $x = P^T y_2$.

Class ComplexSparseCholesky realizes all four steps by algorithms described in George and Liu (1981). Especially, step one, is a realization of a minimum degree ordering algorithm. The numerical factorization in its standard form is based on a sparse compressed storage scheme. Alternatively, a multifrontal method can be used. The multifrontal method requires more storage but will be faster than the standard method in certain cases. The multifrontal method is based on the routines in Liu (1987). For a detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987) et al. (1987), and Liu (1986, 1989, 1992). The numerical factorization method can be specified by using the NumericFactorizationMethod (p. 89) property.

The Solve (p. 92) method will compute the symbolic and numeric factorizations if they have not already been computed or supplied by the user through the FactorSymbolically (p. 91), FactorNumerically (p. 91), SetNumericFactor (p. 92), or SetSymbolicFactor (p. 92) methods. These factorizations are retained for later use by the Solve method when different right-hand sides are to be solved.

There is a special situation where computations can be simplified. If an application generates different sparse Hermitian positive definite coefficient matrices that all have the same sparsity pattern, then by using methods GetSymbolicFactor (p. 92) and SetSymbolicFactor (p. 92) the symbolic factorization needs only be computed once.

Properties

LargestDiagonalElement
public double LargestDiagonalElement {get; }

Description
The largest diagonal element of the Cholesky factor.

Property Value
A double containing the largest diagonal element of the Cholesky factor. Use of this method is only sensible if a numeric factorization of the input matrix was done beforehand.

NumberOfNonzeros
public long NumberOfNonzeros {get; }

Description
The number of nonzeros in the Cholesky factor.

Property Value
A long containing the number of nonzeros (including the diagonal) of the Cholesky factor.

NumericFactorizationMethod
public Imsl.Math.ComplexSparseCholesky.NumericFactorization NumericFactorizationMethod {get; set; }

Linear Systems ComplexSparseCholesky • 89
**Description**

The method used in the numerical factorization of the permuted input matrix.

**Property Value**

An int value equal to `StandardMethod` (p. 87) or `MultiFrontalMethod` (p. 87) representing the method used during the numeric factorization phase:

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>StandardMethod</code> (p. 94)</td>
<td>standard method as described by George/Liu (1981). This is the default.</td>
</tr>
<tr>
<td><code>MultiFrontalMethod</code> (p. 94)</td>
<td>multifrontal method</td>
</tr>
</tbody>
</table>

**Exception**

`System.ArgumentException` is thrown when the value for `NumericFactorizationMethod` is not `NumericFactorization.StandardMethod` or `NumericFactorization.MultiFrontalMethod`.

**SmallestDiagonalElement**

```csharp
public double SmallestDiagonalElement {get; }
```

**Description**

The smallest diagonal element of the Cholesky factor.

**Property Value**

A double containing the smallest diagonal element of the Cholesky factor. Use of this method is only sensible if a numeric factorization of the input matrix was done beforehand.

**Constructor**

```csharp
public ComplexSparseCholesky(Imsl.Math.ComplexSparseMatrix A)
```

**Description**

Constructs the matrix structure for the Cholesky factorization of a sparse Hermitian positive definite matrix of type `ComplexSparseMatrix`.

**Parameter**

A – The `ComplexSparseMatrix` Hermitian positive definite matrix to be factored. Only the lower triangular part of the input matrix is used.
Methods

**FactorNumerically**

```java
public void FactorNumerically()
```

**Description**

Computes the numeric factorization of a sparse Hermitian positive definite matrix.

**Remarks**

This method numerically factors the instance of the constructed matrix $A$, where $A$ is of type `ComplexSparseMatrix` and is Hermitian positive definite. The factorization is obtained in several steps:

1. First, matrix $A$ is permuted to reduce fill-in, leading to a sparse Hermitian positive definite matrix $PAP^T$.
2. Then, matrix $PAP^T$ is symbolically and numerically factored.

Note that the symbolic factorization is not done if the symbolic factor has been supplied by the user through the `SetSymbolicFactor` method.

**Exception**

`Imsl.Math.NotSPDEException` is thrown if the input matrix is not Hermitian, positive definite.

**FactorSymbolically**

```java
public void FactorSymbolically()
```

**Description**

Computes the symbolic factorization of a sparse Hermitian positive definite matrix.

**Remarks**

This method symbolically factors the instance of the constructed matrix $A$, where $A$ is of type `ComplexSparseMatrix` and is Hermitian positive definite. The factorization is obtained in several steps:

1. First, matrix $A$ is permuted to reduce fill-in, leading to a sparse Hermitian positive definite matrix $PAP^T$.
2. Then, matrix $PAP^T$ is symbolically factored.

**Exception**

`Imsl.Math.NotSPDEException` is thrown if the input matrix is not Hermitian, positive definite.

**GetNumericFactor**

```java
```

**Description**

Returns the numeric Cholesky factor.
Returns
A NumericFactor containing the numeric Cholesky factor.

GetSymbolicFactor

Description
Returns the symbolic Cholesky factor.

Returns
A SymbolicFactor containing the symbolic Cholesky factor.

SetNumericFactor

Description
Sets the numeric Cholesky factor to use in solving a sparse complex Hermitian positive definite system of linear equations $Ax = b$.

Parameter
numericFactor – A NumericFactor containing the numeric Cholesky factor. By default the numeric factorization is computed.

SetSymbolicFactor
public void SetSymbolicFactor(Imsl.Math.ComplexSparseCholesky.SymbolicFactor symbolicFactor)

Description
Sets the symbolic Cholesky factor to use in solving a sparse complex Hermitian positive definite system of linear equations $Ax = b$.

Parameter
symbolicFactor – a SymbolicFactor containing the symbolic Cholesky factor. By default the symbolic factorization is computed.

Solve

Description
Computes the solution of a sparse Hermitian positive definite system of linear equations $Ax = b$.

Parameter
b – A Complex vector of length equal to the order of matrix $A$ containing the right-hand side.

Returns
A Complex vector of length equal to the order of matrix $A$ containing the solution of the system $Ax = b$. 
Remarks

This method solves the linear system $Ax = b$, where $A$ is Hermitian positive definite. The solution is obtained in several steps:

1. First, matrix $A$ is permuted to reduce fill-in, leading to a sparse Hermitian positive definite system $PAP^T (Px) = Pb$.
2. Then, matrix $PAP^T$ is symbolically and numerically factored.
3. The final solution is obtained by solving the systems $Ly_1 = Pb, L^T y_2 = y_1$ and $x = P^T y_2$.

By default this method implements all of the above steps. The factorizations are retained for later use by subsequent solves. By choosing appropriate methods within this class, the computation can be reduced to the solution of the system $Ax = b$ for a given or precomputed symbolic or numeric factor.

Exception

`Imsl.Math.NotSPDException` is thrown when the input matrix is not Hermitian, positive definite.

Example: Complex Sparse Cholesky Factorization

The Cholesky Factorization of a sparse Hermitian positive definite matrix is computed. Some additional information about the Cholesky factorization is also computed.

```csharp
using System;
using Imsl.Math;

public class ComplexSparseCholeskyEx1
{
    public static void Main(String[] args)
    {
        ComplexSparseMatrix A = new ComplexSparseMatrix(3, 3);
        A.Set(0, 0, new Complex(2.0, 0.0));
        A.Set(1, 0, new Complex(-1.0, -1.0));
        A.Set(1, 1, new Complex(4.0, 0.0));
        A.Set(2, 1, new Complex(1.0, -2.0));
        A.Set(2, 2, new Complex(10.0, 0.0));
        Complex[] b = { new Complex(-2.0, 2.0), new Complex(5.0, 15.0),
                        new Complex(36.0, 28.0)};
        ComplexSparseCholesky cholesky = new ComplexSparseCholesky(A);
        // Choose Multifrontal method as numeric factorization method
        cholesky.NumericFactorizationMethod =
            ComplexSparseCholesky.NumericFactorization.MultiFrontalMethod;
        // Compute solution
        Complex[] solution = cholesky.Solve(b);
        PrintMatrix p = new PrintMatrix("Computed solution");
        p.Print(solution);
    }
}
```

Linear Systems ComplexSparseCholesky • 93
// Compute additional information about the factorization

Console.Out.Write("Smallest diagonal element of Cholesky factor: ");
Console.Out.WriteLine(cholesky.SmallestDiagonalElement);
Console.Out.Write("Largest diagonal element of Cholesky factor: ");
Console.Out.WriteLine(cholesky.LargestDiagonalElement);
Console.Out.Write("Number of nonzeros in Cholesky factor: ");
Console.Out.WriteLine(cholesky.NumberOfNonzeros);

Output
Computed solution
  0
  0 1+1i
  1 2+2i
  2 3+3i

Smallest diagonal element of Cholesky factor: 1.4142135623731
Largest diagonal element of Cholesky factor: 2.88675134594813
Number of nonzeros in Cholesky factor: 5

ComplexSparseCholesky.NumericFactorization
Enumeration

Numeric Factorization methods.

Fields

MultiFrontalMethod

Description
Indicates the multifrontal method will be used for numeric factorization.

StandardMethod

Description
Indicates that the method of George/Liu (1981) is used for numeric factorization.
ComplexSparseCholesky.NumericFactor Class

public class Imsl.Math.ComplexSparseCholesky.NumericFactor

Data structures and functions for the numeric Cholesky factor.

Used by GetNumericFactor and SetNumericFactor to hold the numeric Cholesky factorization of a matrix.

ComplexSparseCholesky.SymbolicFactor Class

public class Imsl.Math.ComplexSparseCholesky.SymbolicFactor

Data structures and functions for the symbolic Cholesky factor.

Used by GetSymbolicFactor and SetSymbolicFactor to hold the symbolic Cholesky factorization of a matrix.

QR Class

public class Imsl.Math.QR

QR Decomposition of a matrix.

Class QR computes the $QR$ decomposition of a matrix using Householder transformations. It is based on the LINPACK routine SQRDC; see Dongarra et al. (1979).

QR determines an orthogonal matrix $Q$, a permutation matrix $P$, and an upper trapezoidal matrix $R$ with diagonal elements of nonincreasing magnitude, such that $AP = QR$. The Householder transformation for column $k$ is of the form

$$I - \frac{u_k u_k^T}{P_k}$$

for $k = 1, 2, \ldots, \min(\text{number of rows of } A, \text{number of columns of } A)$, where $u$ has zeros in the first $k - 1$ positions. The matrix $Q$ is not produced directly by QR. Instead the information needed to reconstruct the Householder transformations is saved. If the matrix $Q$ is needed explicitly, use the $Q$ property. This method accumulates $Q$ from its factored form.
Before the decomposition is computed, initial columns are moved to the beginning of the array A and the final columns to the end. Both initial and final columns are frozen in place during the computation. Only free columns are pivoted. Pivoting is done on the free columns of largest reduced norm.

Property

**NumberOfProcessors**

public int NumberOfProcessors {get; set; }

**Description**

Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**

An int indicating the maximum possible number of processors to use.

**Remarks**

By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

Constructor

**QR**

public QR(double[,] a)

**Description**

Constructs the QR decomposition of a matrix with elements of type double.

**Parameter**

a – A double matrix to be factored.

Methods

**GetPermute**

public int[] GetPermute()

**Description**

Returns an int array containing information about the permutation of the elements of the matrix during pivoting.
Returns
The $k$-th element contains the index of the column of the matrix that has been interchanged into the $k$-th column.

GetQ
public double[,] GetQ()

Description
The orthogonal or unitary matrix Q.

Returns
A double matrix containing the accumulated orthogonal matrix Q from the QR decomposition.

GetR
public double[,] GetR()

Description
The upper trapezoidal matrix R.

Returns
The upper trapezoidal double matrix R of the QR decomposition.

GetRank
public int GetRank()

Description
Returns the rank of the matrix used to construct this instance.

Returns
An int specifying the rank of the matrix used to construct this instance.

GetRank
double tolerance)

Description
Returns the rank of the matrix given an input tolerance.

Parameter
tolerance – A double scalar value used in determining the rank of the matrix.

Returns
An int specifying the rank of the matrix.

Solve
public double[] Solve(double[] b)

Description
Returns the solution to the least-squares problem $Ax = b$. 
Parameter

  b – A double array to be manipulated.

Returns

A double array containing the solution vector to $Ax = b$ with components corresponding to the unused columns set to zero.

Exception

Imsl.Math.SingularMatrixException is thrown when the upper triangular matrix $R$ resulting from the QR factorization is singular

**Solve**

public double[] Solve(double[] b, double tol)

Description

Returns the solution to the least-squares problem $Ax = b$ using an input tolerance.

Parameters

  b – A double array to be manipulated.
  tol – A double scalar value used in determining the rank of $A$.

Returns

A double array containing the solution vector to $Ax = b$ with components corresponding to the unused columns set to zero.

Exception

Imsl.Math.SingularMatrixException is thrown when the upper triangular matrix $R$ resulting from the QR factorization is singular

**Example: QR Factorization of a Matrix**

The QR Factorization of a Matrix is performed. A linear system is then solved using the factorization. The rank of the input matrix is also computed.

```csharp
using System;
using Imsl.Math;

public class QREx1
{
    public static void Main(String[] args)
    {
        double[,] a = {
            {1, 2, 4},
            {1, 4, 16},
            {1, 6, 36},
            {1, 8, 64}
        };
        double[] b = new double[4]{16.99, 57.01, 120.99, 209.01};
```
// Compute the QR factorization of A
QR qr = new QR(a);

// Solve Ax = b
double[] x = qr.Solve(b);
new PrintMatrix("x").Print(x);

// Print Q and R.
new PrintMatrix("Q").Print(qr.GetQ());
new PrintMatrix("R").Print(qr.GetR());

// Find the rank of A.
int rank = qr.GetRank();
Console.Out.WriteLine("rank = " + rank);
}

Output

x
0 0.990000000000019
1 2.00199999999999
2 3

Q
0 -0.0531494003452735 -0.54217094609664 0.808223859120487 -0.22360679774998
1 -0.212597601381094 -0.657435635424271 -0.269407953040162 0.670820393249937
2 -0.478344603107461 -0.345794067982896 -0.449013255066938 -0.670820393249936
3 -0.850390405524374 0.392753756227487 0.269407953040163 0.223606797749979

R
0 -75.2595508889071 -10.6298800690547 -1.5944820103582
1 0 -2.64681879196785 -1.15264689327632
2 0 0 -1.15264689327632
3 0 0 0

rank = 3

SVD Class

class SVD

Singular Value Decomposition (SVD) of a rectangular matrix of type double.

SVD is based on the LINPACK routine SSVDC; see Dongarra et al. (1979).

Let n be the number of rows in A and let p be the number of columns in A. For any
In an $n \times p$ matrix $A$, there exists an $n \times n$ orthogonal matrix $U$ and a $p \times p$ orthogonal matrix $V$ such that

$$U^T A V = \begin{cases} 
\begin{bmatrix} \Sigma \\ 0 \\ 0 \end{bmatrix} & \text{if } n \geq p \\
\begin{bmatrix} 0 \\
\Sigma \\
0 \end{bmatrix} & \text{if } n \leq p
\end{cases}$$

where $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_m)$, and $m = \min(n, p)$. The scalars $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m \geq 0$ are called the singular values of $A$. The columns of $U$ are called the left singular vectors of $A$. The columns of $V$ are called the right singular vectors of $A$.

The estimated rank of $A$ is the number of $\sigma_k$ that is larger than a tolerance $\eta$. If $\tau$ is the parameter $\text{tol}$ in the program, then

$$\eta = \begin{cases} 
\tau & \text{if } \tau > 0 \\
|\tau| \|A\|_{\infty} & \text{if } \tau < 0
\end{cases}$$

The Moore-Penrose generalized inverse of the matrix is computed by partitioning the matrices $U$, $V$ and $\Sigma$ as $U = (U_1, U_2)$, $V = (V_1, V_2)$ and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_k)$ where the “1” matrices are $k \times k$. The Moore-Penrose generalized inverse is $V_1 \Sigma_1^{-1} U_1^T$.

**Properties**

**Info**

public int Info {get; }

**Description**

Returns the index of the first singular value for which the algorithm converged.

**Property Value**

Convergence was obtained for the Info, Info+1, ..., Min(nrows,ncols) singular values and their corresponding vectors. Here, nrows and ncols represent the number of rows and columns of the input matrix respectively.

**NumberOfProcessors**

public int NumberOfProcessors {get; set; }

**Description**

Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**

An int indicating the maximum possible number of processors to use.
Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

Rank
public int Rank {get; }

Description
Returns the rank of the matrix used to construct this instance.

Property Value
An int scalar containing the rank of the matrix used to construct this instance. The estimated rank of the input matrix is the number of singular values which are larger than a tolerance.

Constructors

SVD
public SVD(double[,] a, double tol)

Description
Construct the singular value decomposition of a rectangular matrix with a given tolerance.

Parameters
  a – A double matrix for which the singular value decomposition is to be computed.
  tol – A double scalar containing the tolerance used to determine when a singular value is negligible.

Remarks
If tol is positive, then a singular value is considered negligible if the singular value is less than or equal to tol. If tol is negative, then a singular value is considered negligible if the singular value is less than or equal to the absolute value of the product of tol and the infinity norm of the input matrix. In the latter case, the absolute value of tol generally contains an estimate of the level of the relative error in the data.

Exception
  ImSL.Math.DidNotConvergeException is thrown when the rank cannot be determined because convergence was not obtained for all singular values.

SVD
public SVD(double[,] a)

Description
Construct the singular value decomposition of a rectangular matrix with default tolerance.

Linear Systems
**Parameter**

- `a` – A `double` matrix for which the singular value decomposition is to be computed.

**Remarks**

The tolerance used is $2.2204460492503\times10^{-14}$. This tolerance is used to determine rank. A singular value is considered negligible if the singular value is less than or equal to this tolerance.

**Methods**

---

**GetS**

```csharp
public double[] GetS()
```

**Description**

Returns the singular values.

**Returns**

A `double` array containing the singular values of the matrix.

---

**GetU**

```csharp
public double[,] GetU()
```

**Description**

Returns the left singular vectors.

**Returns**

A `double` matrix containing the left singular vectors.

---

**GetV**

```csharp
public double[,] GetV()
```

**Description**

Returns the right singular vectors.

**Returns**

A `double` matrix containing the right singular vectors.

---

**Inverse**

```csharp
public double[,] Inverse()
```

**Description**

Compute the Moore-Penrose generalized inverse of a real matrix.

**Returns**

A `double` matrix containing the generalized inverse of the matrix used to construct this instance.
Example: Singular Value Decomposition of a Matrix

The singular value decomposition of a matrix is performed. The rank of the matrix is also computed.

```csharp
using System;
using Imsl.Math;

public class SVDEx1
{
    public static void Main(String[] args)
    {
        double[,] a = {
            {1, 2, 1, 4},
            {3, 2, 1, 3},
            {4, 3, 1, 4},
            {2, 1, 3, 1},
            {1, 5, 2, 2},
            {1, 2, 2, 3}
        };

        // Compute the SVD factorization of A
        SVD svd = new SVD(a);

        // Print U, S and V.
        new PrintMatrix("U").SetPageWidth(80).Print(svd.GetU());
        new PrintMatrix("S").SetPageWidth(80).Print(svd.GetS());
        new PrintMatrix("V").SetPageWidth(80).Print(svd.GetV());

        // Find the rank of A.
        int rank = svd.Rank;
        Console.Out.WriteLine("rank = " + rank);
    }
}
```

Output

```
 U
0  0.119670992640587  0.119670992640587  0.439082824383239
1 -0.380475586320569 -0.380475586320569 -0.565399585908374
2 -0.545120486248343 -0.545120486248343 -0.54628656614145
3 -0.264784294004146 -0.264784294004146 -0.354628656614145
4 -0.403753713172442 -0.403753713172442 -0.446310112301556
5 -0.545800221853259 -0.545800221853259 -0.55399585908374

 S
0
```

Linear Systems
GenMinRes Class

```java
public class Imsl.Math.GenMinRes

Linear system solver using the restarted Generalized Minimum Residual (GMRES) method.

GenMinRes implements restarted GMRES to generate an approximate solution to $Ax = b$. It is based on
GMRES by Homer Walker (1988).

The GMRES method begins with an approximate solution $x_0$ and an initial residual $r_0 = b - Ax_0$. At
iteration $m$, a correction $z_m$ is determined in the Krylov subspace
$$
\kappa_m(v) = \text{span}(v, Av, \ldots, A^{m-1}v)
$$
$v = r_0$ which solves the least squares problem
$$
\min_{z \in \kappa_m(r_0)} \| b - A(x_0 + z) \|_2
$$
Then at iteration $m$, $x_m = x_0 + z_m$.

There are four distinct GMRES implementations, selectable through property `Method`. The first
Gram-Schmidt implementation is essentially the original algorithm by Saad and Schultz (1986). The
second Gram-Schmidt implementation, developed by Homer Walker and Lou Zhou, is simpler than the
first implementation. The least squares problem is constructed in upper-triangular form and the residual
vector updating at the end of a GMRES cycle is cheaper. The first Householder implementation is
algorithm 2.2 of Walker (1988), but with more efficient correction accumulation at the end of each
GMRES cycle. The second Householder implementation is algorithm 3.1 of Walker (1988). The
products of Householder transformations are expanded as sums, allowing most work to be formulated as
large scale matrix-vector operations.

104 • GenMinRes  IMSG C# Numerical Library
The Gram-Schmidt implementations are less expensive than the Householder, the latter requiring about twice as many computations beyond the coefficient matrix/vector products. However, the Householder implementations may be more reliable near the limits of residual reduction. See Walker (1988) for details. Issues such as the cost of coefficient matrix/vector products, availability of effective preconditioners, and features of particular computing environments may serve to mitigate the extra expense of the Householder implementations.

**Properties**

**Iterations**

```csharp
public int Iterations {get; }
```

**Description**
The actual number of GMRES iterations used.

**Property Value**
An `int` scalar representing the number of iterations used.

**MaxIterations**

```csharp
public int MaxIterations {get; set; }
```

**Description**
The maximum number of iterations allowed.

**Property Value**
An `int` specifying the maximum number of iterations allowed.

**Remarks**
By default, `MaxIterations` = 1000.

**Exception**
`System.ArgumentException` is thrown if `MaxIterations` is less than or equal to 0.

**MaxKrylovDim**

```csharp
public int MaxKrylovDim {get; set; }
```

**Description**
The maximum Krylov subspace dimension.

**Property Value**
An `int` scalar representing the maximum Krylov subspace dimension, that is, the maximum allowable number of GMRES iterations allowed before restarting.

**Remarks**
By default, `MaxKrylovDim` is set to the minimum of `n` and 20 where `n` is the order of the system to be solved.
Exception

System.ArgumentException is thrown if MaxKrylovDim is less than 1 or greater than n.

Method

public Imsl.Math.GenMinRes.ImplementationMethod Method {get; set; }

Description

The implementation method to be used.

Property Value

An ImplementationMethod value specifying the implementation method to be used.

Remarks

<table>
<thead>
<tr>
<th>ImplementationMethod</th>
<th>Method Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstGramSchmidt</td>
<td>Use the first Gram-Schmidt implementation. This is the default value used.</td>
</tr>
<tr>
<td>SecondGramSchmidt</td>
<td>Use the second Gram-Schmidt implementation.</td>
</tr>
<tr>
<td>FirstHouseholder</td>
<td>Use the first Householder implementation.</td>
</tr>
<tr>
<td>SecondHouseholder</td>
<td>Use the second Householder implementation.</td>
</tr>
</tbody>
</table>

By default, Method = GenMinRes.ImplementationMethod.FirstGramSchmidt.

PreconditionerSolves

public int PreconditionerSolves {get; }

Description

The total number of GMRES right preconditioner solves.

Property Value

An int representing the number of GMRES right preconditioner solves.

Products

public int Products {get; }

Description

The total number of GMRES matrix-vector products used.

Property Value

An int representing the number of GMRES matrix-vector products used.

RelativeError

public double RelativeError {get; set; }

Description

The stopping tolerance.
Property Value
A double scalar value specifying the stopping tolerance.

Remarks
The algorithm attempts to generate $x$ such that $\|b - Ax\|_2 \leq t \|b\|_2$, where $t = \text{RelativeError}$. By default, RelativeError is set to $1.4901161193847656e-08$.

Exception
System.ArgumentException is thrown if RelativeError is less than or equal to 0.0.

ResidualNorm
public double ResidualNorm {get; }

Description
The final residual norm, $\|b - Ax\|_2$.

Property Value
A double scalar value specifying the final residual norm.

ResidualUpdating
public Imsl.Math.GenMinRes.ResidualMethod ResidualUpdating {get; set; }

Description
The residual updating method to be used.

Property Value
A ResidualMethod value specifying the residual updating method to be used.

Remarks

<table>
<thead>
<tr>
<th>ResidualMethod</th>
<th>Updating Method Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinearAtRestartOnly</td>
<td>Update by linear combination upon restarting only. This is the default value used.</td>
</tr>
<tr>
<td>LinearAtRestartAndTermination</td>
<td>Update by linear combination upon restarting and at termination.</td>
</tr>
<tr>
<td>DirectAtRestartOnly</td>
<td>Update by direct evaluation upon restarting only.</td>
</tr>
<tr>
<td>DirectAtRestartAndTermination</td>
<td>Update by direct evaluation upon restarting and at termination.</td>
</tr>
</tbody>
</table>

By default, ResidualUpdating is set to LinearAtRestartOnly.

Constructor

GenMinRes
public GenMinRes(int n, Imsl.Math.GenMinRes.IFunction argF)
Description
GMRES linear system solver constructor.

Parameters
n – An int scalar value which defines the order of the system to be solved.
argF – An IFunction that defines the user-supplied function which computes \( \mathbf{z} = A\mathbf{p} \). If argF implements IPreconditioner then right preconditioning is performed using this user supplied function. Otherwise, no preconditioning is performed. Note that argF can be used to act upon the coefficients of matrix A stored in different storage modes. See the examples.

Methods

GetGuess
public double[] GetGuess()

Description
Returns the initial guess of the solution.

Returns
A double array of length n containing the initial guess of the solution.

GetVectorProducts
public Imsl.Math.GenMinRes.IVectorProducts GetVectorProducts()

Description
Returns the user-supplied functions for the inner product and, optionally, the norm used in the Gram-Schmidt implementations.

Returns
An IVectorProducts that defines the user-supplied functions for the inner product and, optionally, the norm used in the Gram-Schmidt implementations.

SetGuess
public void SetGuess(double[] xGuess)

Description
Sets the initial guess of the solution.

Parameter
xGuess – A double array of length n containing the initial guess of the solution.

Remarks
By default the elements of this array are set to 0.0.

SetVectorProducts
public void SetVectorProducts(Imsl.Math.GenMinRes.IVectorProducts argP)
Description
Sets the user-supplied functions for the inner product and, optionally, the norm to be used in the
Gram-Schmidt implementations.

Parameter
argP – An IVectorProducts specifying the user-defined function for the inner product and,
optionally, the norm in the Gram-Schmidt implementations. If this member function is not called,
the dot product will be used for the inner product and the $L_2$ norm will be used for the norm.

Solve
public double[] Solve(double[] b)

Description
Generate an approximate solution to $Ax = b$ using the Generalized Residual Method.

Parameter
b – A double array which defines the right-hand side of the linear system.

Returns
A double array containing the solution of the linear system.

Exception
System.ArgumentException is thrown if the length of b is not consistent with n.

Example 1: Solve a Small Linear System
A solution to a small linear system is found. The coefficient matrix is stored as a full matrix and no
preconditioning is used. Typically, preconditioning is required to achieve convergence in a reasonable
number of iterations.

using System;
using Imsl.Math;
using IMSLException = Iml.IMSLException;

public class GenMinResEx1 : GenMinRes.IFunction
{
    private static double[,] a = {
        {33.0, 16.0, 72.0},
        {-24.0, -10.0, -57.0},
        {18.0, -11.0, 7.0}
    };
    private static double[] b = {129.0, -96.0, 8.5};
    // If A were to be read in from some outside source the  //
    // code to read the matrix could reside in a constructor.  //

    public void Amultp(double[] p, double[] z)
    {
        double[] result;
        result = Matrix.Multiply(a, p);
        Array.Copy(result, 0, z, 0, z.Length);
    }
}
public static void Main(String[] args) {
    int n = 3;
    GenMinResEx1 atp = new GenMinResEx1();
    // Construct a GenMinRes object
    GenMinRes gnmnrs = new GenMinRes(n, atp);
    // Solve Ax = b
    new PrintMatrix("x").Print(gnmnrs.Solve(b));
}

Output

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.999999999999999</td>
</tr>
<tr>
<td>1</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Example 2: Solve a Small Linear System with User Supplied Inner Product

A solution to a small linear system is found. The coefficient matrix is stored as a full matrix and no preconditioning is used. Typically, preconditioning is required to achieve convergence in a reasonable number of iterations. The user supplies a function to compute the inner product and norm within the Gram-Schmidt implementation.

using System;
using Imsl.Math;
using IMSLException = Imsl.IMSLException;

public class GenMinResEx2 : GenMinRes.IFunction, GenMinRes.IVectorProducts {
    private static double[,] a = {
        {33.0, 16.0, 72.0},
        {-24.0, -10.0, -57.0},
        {18.0, -11.0, 7.0}
    };
    private static double[] b = {129.0, -96.0, 8.5};
    // If A were to be read in from some outside source the //
    // code to read the matrix could reside in a constructor. //
    public void Amultp(double[] p, double[] z) {
        double[] result;
        result = Matrix.Multiply(a, p);
        Array.Copy(result, 0, z, 0, z.Length);
    }
}
```csharp
public double Innerproduct(double[] x, double[] y)
{
    int n = x.Length;
    double tmp = 0.0;
    for (int i = 0; i < n; i++)
    {
        tmp += x[i] * y[i];
    }
    return tmp;
}

public double Norm(double[] x)
{
    int n = x.Length;
    double tmp = 0.0;
    for (int i = 0; i < n; i++)
    {
        tmp += x[i] * x[i];
    }
    return System.Math.Sqrt(tmp);
}

public static void Main(String[] args)
{
    int n = 3;
    GenMinResEx2 atp = new GenMinResEx2();

    // Construct a GenMinRes object
    GenMinRes gnmnrs = new GenMinRes(n, atp);
    gnmnrs.SetVectorProducts(atp);

    // Solve Ax = b
    new PrintMatrix("x").Print(gnmnrs.Solve(b));
}
}
```

**Output**

```
x
0
0 0.999999999999999
1 1.5
2 1
```

**Example 3: Solve a Small Linear System Stored in Sparse Form**

A solution to a small linear system in which the coefficient matrix has been stored in SparseMatrix form is found. An initial guess of ones is set before solving the system.

```csharp
using System;

Linear Systems GenMinRes • 111
```
using Imsl.Math;
public class GenMinResEx3 : GenMinRes.IFunction
{
    private static SparseMatrix A;
    private static double[] a = {6.0, 10.0, 15.0, -3.0, 10.0, -1.0,
                                   -1.0, -3.0, -5.0, 1.0, 10.0, -1.0,
                                   -2.0, 1.0, 0.0, -2.0};
    private static int[] irow = {5, 1, 2, 1, 3, 3, 4, 4, 4, 5, 1, 3};
    private static int[] jcol = {5, 1, 2, 2, 3, 4, 0, 5, 3, 4, 0, 0};
    private static double[] b = {10.0, 7.0, 45.0, 33.0, -34.0, 31.0};
    private static double[] xguess = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0};

    public void Amultp(double[] p, double[] z)
    {
        double[] result;
        result = Imsl.Math.SparseMatrix.Multiply(A, p);
        Array.Copy(result, 0, z, 0, z.Length);
    }

    public static void Main(String[] args)
    {
        int n = 6;
        A = new SparseMatrix(n, n);
        for (int i = 0; i < a.Length; i++)
        {
            A.Set(irow[i], jcol[i], a[i]);
        }
        GenMinResEx3 atp = new GenMinResEx3();

        // Construct a GenMinRes object
        GenMinRes gnmnrs = new GenMinRes(n, atp);
        gnmnrs.SetGuess(xguess);
        // Solve Ax = b
        new PrintMatrix("x").Print(gnmnrs.Solve(b));
    }
}

Output

<table>
<thead>
<tr>
<th></th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Example 4: Solve a Small Linear System Stored in Sparse Form With Preconditioning

A solution to a small linear system in which the coefficient matrix has been stored in SparseMatrix form is found. An initial guess of ones is set before solving the system and preconditioning is used.

```csharp
using System;
using Imsl.Math;
public class GenMinResEx4 : GenMinRes.IPreconditioner
{
    private static SparseMatrix A;
    private static double[] a = {6.0, 10.0, 15.0, -3.0, 10.0, -1.0,
                                -1.0, -3.0, -5.0, 1.0, 10.0, -1.0,
                                -2.0, -1.0, -2.0};
    private static double[] b = {10.0, 7.0, 45.0, 33.0, -34.0, 31.0};
    private static double[] xguess = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
    private static double[] diagin = {0.1, 0.1, 0.066666666666667, 0.1,
                                      1.0, 0.1666666666666667};
    private static int[] irow = {5, 1, 2, 1, 3, 4, 4, 4, 4, 4, 0, 5, 5, 1, 3};
    private static int[] jcol = {5, 1, 2, 2, 3, 4, 0, 5, 3, 4, 0, 0, 1, 3, 0};

    public virtual void Amultp(double[] p, double[] z)
    {
        double[] result;
        result = Imsl.Math.SparseMatrix.Multiply(A, p);
        Array.Copy(result, 0, z, 0, z.Length);
    }

    public void Preconditioner(double[] r, double[] z)
    {
        int n = z.Length;
        for (int i = 0; i < n; i++)
        {
            z[i] = diagin[i] * r[i];
        }
    }

    public static void Main(String[] args)
    {
        int n = 6;
        A = new SparseMatrix(n, n);
        for (int i = 0; i < a.Length; i++)
        {
            A.Set(irow[i], jcol[i], a[i]);
        }
        GenMinResEx4 atp = new GenMinResEx4();
        // Construct a GenMinRes object
        GenMinRes gnmnrs = new GenMinRes(n, atp);
        gnmnrs.SetGuess(xguess);
        // Solve Ax = b
        new PrintMatrix("x").Print(gnmnrs.Solve(b));
    }
}
```
Example 5: The Second Householder Implementation

The coefficient matrix in this example corresponds to the five-point discretization of the 2-d Poisson equation with the Dirichlet boundary condition. Assuming the natural ordering of the unknowns, and moving all boundary terms to the right hand side, we obtain a block tridiagonal matrix. (Consider the tridiagonal matrix $T$ which has the value 4.0 down the main diagonal and -1.0 along the upper and lower co-diagonals. Then the coefficient matrix is the block tridiagonal matrix consisting of $T$'s down the main diagonal and -I along the upper and lower codiagonals where I is the identity matrix.) Discretizing on a 20 x 20 grid implies that the coefficient matrix is 400 x 400. In the solution, the second Householder implementation is selected and we choose to update the residual vector by direct evaluation.

```csharp
using System;
using Imsl.Math;
public class GenMinResEx5 : GenMinRes.IFunction
{
    // Creates a new instance of GenMinResEx5
    public GenMinResEx5()
    {
    }
    public void Amultp(double[] p, double[] z)
    {
        int n = z.Length;
        int k = (int) Math.Sqrt(n);
        // Multiply by diagonal blocks
        for (int i = 0; i < n; i++)
        {
            z[i] = 4.0 * p[i];
        }
        for (int i = 0; i < n - 2; i++)
        {
            z[i] = (-1.0) * p[i + 1] + z[i];
        }
        for (int i = 0; i < n - 2; i++)
        {
            z[i + 1] = (-1.0) * p[i] + z[i + 1];
        }
        // Correct for terms not properly in block diagonal
        for (int i = k - 1; i < n - k; i = i + k)
        {
            z[i] += p[i + 1];
            z[i + 1] += p[i];
        }
    }
```
// Do the super and subdiagonal blocks, the -I's
for (int i = 0; i < n - k; i++)
{
    z[i] = (-1.0) * p[i + k] + z[i];
}
for (int i = 0; i < n - k; i++)
{
    z[i + k] = (-1.0) * p[i] + z[i + k];
}

public static void Main(String[] args)
{
    int n = 400;
    double[] b = new double[n];
    double[] xguess = new double[n];
    GenMinResEx5 atp = new GenMinResEx5();
    // Construct a GenMinRes object
    GenMinRes gnmnrs = new GenMinRes(n, atp);
    // Set right hand side and initial guess to ones */
    for (int i = 0; i < n; i++)
    {
        b[i] = 1.0;
        xguess[i] = 1.0;
    }
    gnmnrs.SetGuess(xguess);
    // Solve Ax = b
    gnmnrs.Solve(b);
    int iterations = gnmnrs.Iterations;
    Console.Out.WriteLine("The number of iterations used = " + iterations);
    double resnorm = gnmnrs.ResidualNorm;
    Console.Out.WriteLine("The final residual norm is " + resnorm);
}

Output
The number of iterations used = 92
The final residual norm is 2.52648529541037E-07

Example 6: The Second Householder Implementation With Preconditioning

The coefficient matrix in this example corresponds to the five-point discretization of the 2-d Poisson equation with the Dirichlet boundary condition. Assuming the natural ordering of the unknowns, and moving all boundary terms to the right hand side, we obtain a block tridiagonal matrix. (Consider the tridiagonal matrix T which has the value 4.0 down the main diagonal and -1.0 along the upper and lower
co-diagonals. Then the coefficient matrix is the block tridiagonal matrix consisting of $T$'s down the main diagonal and -1 along the upper and lower co-diagonals where $I$ is the identity matrix.) Discretizing on a 20 x 20 grid implies that the coefficient matrix is 400 x 400. In the solution, the second Householder implementation is selected and we choose to update the residual vector by direct evaluation. Preconditioning is used with the preconditioning matrix being a diagonal matrix with 4.0 down the main diagonal and -1.0 along the upper and lower co-diagonals. This Preconditioner method solves this tridiagonal matrix.

```
using System;
using Imsl.Math;
public class GenMinResEx6 : GenMinRes.IPreconditioner
{
    private double[] precondA, precondB, precondC;

    // Creates a new instance of GenMinResEx6
    public GenMinResEx6(int n)
    {
        precondA = new double[n];
        precondB = new double[n];
        precondC = new double[n];
        // Define the preconditioning matrix
        for (int i = 0; i < n; i++)
        {
            precondA[i] = 4.0;
            precondB[i] = -1.0;
            precondC[i] = -1.0;
        }
    }

    public void Amultp(double[] p, double[] z)
    {
        int m = z.Length;
        int n = (int) Math.Sqrt(m);
        // Multiply by diagonal blocks
        for (int i = 0; i < m; i++)
        {
            z[i] = 4.0 * p[i];
        }
        for (int i = 0; i < m - 2; i++)
        {
            z[i] -= p[i + 1];
        }
        for (int i = 0; i < m - 2; i++)
        {
            z[i + 1] -= p[i];
        }
        // Correct for terms not properly in block diagonal
        for (int i = n - 1; i < m - n; i += n)
        {
            z[i] += p[i + 1];
            z[i + 1] += p[i];
        }
        // Do the super and subdiagonal blocks, the -1's
        for (int i = 0; i < m - n; i++)
    }
```
```csharp
{ 
    z[i] -= p[i + n];
}
for (int i = 0; i < m - n; i++)
{
    z[i + n] -= p[i];
}

/// Solve the tridiagonal preconditioning matrix problem for z.
public void Preconditioner(double[] r, double[] z)
{
    int n = z.Length;
    double[] w = new double[n];
    double[] v = new double[n];
    double[] u = new double[n];
    w[0] = precondA[0];
    v[0] = precondC[0] / w[0];
    u[0] = r[0] / w[0];
    for (int i = 1; i < n; i++)
    {
        w[i] = precondA[i] - precondB[i] * v[i - 1];
        v[i] = precondC[i] / w[i];
        u[i] = (r[i] - precondB[i] * u[i - 1]) / w[i];
    }  
    z[n - 1] = u[n - 1];
    for (int j = n - 2; j >= 0; j--)
    {
        z[j] = u[j] - v[j] * z[j + 1];
    }
}

public static void Main(String[] args)
{
    int n = 400;
    double[] b = new double[n];
    double[] xguess = new double[n];
    GenMinResEx6 atp = new GenMinResEx6(n);
    // Construct a GenMinRes object
    GenMinRes gnmnrs = new GenMinRes(n, atp);
    // Set right hand side and initial guess to ones
    for (int i = 0; i < n; i++)
    {
        b[i] = 1.0;
        xguess[i] = 1.0;
    }
    gnmnrs.SetGuess(xguess);
    // Solve Ax = b
    gnmnrs.Solve(b);
    int iterations = gnmnrs.Iterations;
    Console.Out.WriteLine("The number of iterations used = " + iterations);
}
```
double resnorm = gnmnrs.ResidualNorm;
Console.Out.WriteLine("The final residual norm is " + resnorm);
}

Output
The number of iterations used = 60
The final residual norm is 2.84141491724154E-07

GenMinRes.ImplementationMethod Enumeration

public enumeration Imsl.Math.GenMinRes.ImplementationMethod
Implementation methods.

Fields

FirstGramSchmidt
public Imsl.Math.GenMinRes.ImplementationMethod FirstGramSchmidt

Description
Indicates the first Gram-Schmidt implementation method is to be used.

FirstHouseholder
public Imsl.Math.GenMinRes.ImplementationMethod FirstHouseholder

Description
Indicates the first Householder implementation method is to be used.

SecondGramSchmidt
public Imsl.Math.GenMinRes.ImplementationMethod SecondGramSchmidt

Description
Indicates the second Gram-Schmidt implementation method is to be used.

SecondHouseholder

Description
Indicates the second Householder implementation method is to be used.
GenMinRes.ResidualMethod Enumeration


Residual updating methods.

Fields

**DirectAtRestartAndTermination**

public Imsl.Math.GenMinRes.ResidualMethod DirectAtRestartAndTermination

**Description**

Indicates residual updating is to be done by direct evaluation upon restarting and at termination.

**DirectAtRestartOnly**

public Imsl.Math.GenMinRes.ResidualMethod DirectAtRestartOnly

**Description**

Indicates residual updating is to be done by direct evaluation upon restarting only.

**LinearAtRestartAndTermination**

public Imsl.Math.GenMinRes.ResidualMethod LinearAtRestartAndTermination

**Description**

Indicates residual updating is to be done by linear combination upon restarting and at termination.

**LinearAtRestartOnly**

public Imsl.Math.GenMinRes.ResidualMethod LinearAtRestartOnly

**Description**

Indicates residual updating is to be done by linear combination upon restarting only.

GenMinRes.IFunction Interface

public interface Imsl.Math.GenMinRes.IFunction

Public interface for the user supplied function to GenMinRes.
**Method**

**Amultp**

abstract public void Amultp(double[] p, double[] z)

**Description**

Used to compute $z = Ap$ where $A$ is the matrix of coefficients to solve and $p$ and $z$ are arrays of length $n$, the order of matrix $A$.

**Parameters**

- $p$ – An input double array of length $n$ generated during the implementation of the Solve method.
- $z$ – An output double array of length $n$.

---

**GenMinRes.IPreconditioner Interface**

public interface Imsl.Math.GenMinRes.IPreconditioner :
Imsl.Math.GenMinRes.IFunction

Public interface for the user supplied function to GenMinRes used for preconditioning.

---

**Method**

**Preconditioner**

abstract public void Preconditioner(double[] r, double[] z)

**Description**

Used to compute $z = M^{-1}r$ where $M$ is the preconditioning matrix and $r$ and $z$ are arrays of length $n$, the order of matrix $M$.

**Parameters**

- $r$ – An input double array of length $n$ generated during the implementation of the Solve method.
- $z$ – An output double array of length $n$.

---

**GenMinRes.IVectorProducts Interface**

public interface Imsl.Math.GenMinRes.IVectorProducts
Public interface for the user supplied function to the GenMinRes object used for the inner product when the Gram-Schmidt implementation is used.

Method

**Innerproduct**

```java
abstract public double Innerproduct(double[] x, double[] y)
```

**Description**

Used to compute the inner product of two vectors for the Gram-Schmidt implementation.

**Parameters**

- `x` – The first input double vector which is to take part in the inner product.
- `y` – The second input double vector which is to take part in the inner product.

**Returns**

A double, the value of the inner product of `x` and `y`.

**Remarks**

If this function is not implemented, the dot product is used for the inner product.

---

**GenMinRes.INorm Interface**

```java
```

Public interface for the user supplied function to the GenMinRes object used for the norm \( \|X\| \) when the Gram-Schmidt implementation is used.

Method

**Norm**

```java
abstract public double Norm(double[] x)
```

**Description**

Used to compute the norm \( \|X\| \) in the Gram-Schmidt implementation.

**Parameter**

- `x` – An input double vector for which the norm will be computed.
Returns
A double, the value of the norm of \( x \).

Remarks
If this function is not implemented, the \( L_2 \) norm is used.

---

**ConjugateGradient Class**

```csharp
public class Imsl.Math.ConjugateGradient

Solves a real symmetric definite linear system using the conjugate gradient method with optional preconditioning.

Class `ConjugateGradient` solves the symmetric positive or negative definite linear system \( Ax = b \) using the conjugate gradient method with optional preconditioning. This method is described in detail by Golub and Van Loan (1983, Chapter 10), and in Hageman and Young (1981, Chapter 7).

The preconditioning matrix \( M \) is a matrix that approximates \( A \), and for which the linear system \( Mz = r \) is easy to solve. These two properties are in conflict; balancing them is a topic of current research. If no preconditioning matrix is specified, \( M \) is set to the identity, i.e. \( M = I \).

The number of iterations needed depends on the matrix and the error tolerance. As a rough guide,

\[ \text{itmax} = \sqrt{n} \text{ for } n \gg 1, \]

where \( n \) is the order of matrix \( A \). See the references for details.

Let \( M \) be the preconditioning matrix, let \( b, p, r, x \) and \( z \) be vectors and let \( \tau \) be the desired relative error. Then the algorithm used is as follows:

- \( \lambda = -1 \)
- \( p_0 = x_0 \)
- \( r_1 = b - Ap_0 \)
- for \( k = 1, \ldots, \text{itmax} \)
  - \( z_k = M^{-1}r_k \)
  - if \( k = 1 \) then
    * \( \beta_k = 1 \)
    * \( p_k = z_k \)
  - else
    * \( \beta_k = (z_k^T r_k) / (z_{k-1}^T r_{k-1}) \)
    * \( p_k = z_k + \beta_k p_{k-1} \)
```

---
\[ \alpha_k = \frac{(r_k^T z_k)}{(p_k^T A p_k)} \]
\[ x_k = x_{k-1} + \alpha_k p_k \]
\[ r_{k+1} = r_k - \alpha_k A p_k \]
\[ \text{if } (\|A p_k\|_2 \leq \tau (1 - \lambda) \|x_k\|_2) \text{ then} \]
\[ \star \text{ recompute } \lambda \]
\[ \star \text{ if } (\|A p_k\|_2 \leq \tau (1 - \lambda) \|x_k\|_2) \text{ exit} \]

Here, \( \lambda \) is an estimate of \( \lambda_{\text{max}}(\Gamma) \), the largest eigenvalue of the iteration matrix \( \Gamma = I - M^{-1} A \). The stopping criterion is based on the result (Hageman and Young 1981, pp. 148-151)
\[ \frac{\|x_k - x\|_M}{\|x\|_M} \leq \left( \frac{1}{1 - \lambda_{\text{max}}(\Gamma)} \right) \frac{\|z_k\|_M}{\|x_k\|_M}, \]
where
\[ \|x\|_M^2 = x^T M x. \]
It is also known that
\[ \lambda_{\text{max}}(T_1) \leq \lambda_{\text{max}}(T_2) \leq \ldots \leq \lambda_{\text{max}}(\Gamma) < 1, \]
where the \( T_l \) are the symmetric, tridiagonal matrices
\[
T_l = \begin{bmatrix}
\mu_1 & \omega_2 \\
\omega_2 & \mu_2 & \omega_3 \\
\omega_3 & \ddots & \ddots & \ddots & \omega_l \\
& \ddots & \ddots & \ddots & \ddots & \omega_l \\
& & \ddots & \ddots & \ddots & \mu_l \\
\end{bmatrix}
\]
with \( \mu_1 = 1 - 1/\alpha_1 \) and, for \( k = 2, \ldots, l, \)
\[ \mu_k = 1 - \beta_k / \alpha_{k-1} - 1 / \alpha_k \quad \text{and} \quad \omega_k = \sqrt{\beta_k / \alpha_{k-1}}. \]

Usually, the eigenvalue computation is needed for only a few of the iterations.

Properties

**Iterations**
public int Iterations { get; }

Linear Systems ConjugateGradient • 123
Description
The number of iterations needed by the conjugate gradient algorithm.

Property Value
An int value indicating the number of iterations needed.

MaxIterations
public int MaxIterations {get; set; }

Description
The maximum number of iterations allowed.

Property Value
An int value specifying the maximum number of iterations allowed.

Remarks
By default, MaxIterations = \max(1000, \sqrt{n})

Exception
System.ArgumentException is thrown if MaxIterations is less than or equal to 0.

RelativeError
public double RelativeError {get; set; }

Description
The relative error used for stopping the algorithm.

Property Value
A double specifying the relative error.

Remarks
By default, RelativeError = 1.49e-08, the square root of the precision.

Exception
System.ArgumentException is thrown if RelativeError is less than 0.

Constructor

ConjugateGradient
public ConjugateGradient(int n, Imsl.Math.ConjugateGradient.IFunction argF)

Description
Conjugate gradient constructor.
Parameters

- **n** – An int scalar value defining the order of the matrix.
- **argF** – An IFunction that defines the user-supplied function which computes \( z = Ap \). If **argF** implements IPreconditioner then right preconditioning is performed using this user supplied function. Otherwise, no preconditioning is performed. Note that **argF** can be used to act upon the coefficients of matrix \( A \) stored in different storage modes.

Methods

**GetJacobi**

```java
public double[] GetJacobi()
```

**Description**

Returns the Jacobi preconditioning matrix.

**Returns**

A `double` vector diagonal containing the diagonal of the Jacobi preconditioner \( M \), that is, \( \text{diagonal}[i]=A_{i,i},A \) the input matrix.

**SetJacobi**

```java
public void SetJacobi(double[] diagonal)
```

**Description**

Defines a Jacobi preconditioner as the preconditioning matrix, that is, \( M \) is the diagonal of \( A \).

**Parameter**

- `diagonal` – A `double` vector containing the diagonal of the Jacobi preconditioner \( M \), that is, \( \text{diagonal}[i]=A_{i,i},A \) the input matrix.

**Exception**

- `System.ArgumentException` is thrown if the length of vector `diagonal` is not equal to the order \( n \) of input matrix \( A \).

**Solve**

```java
public double[] Solve(double[] b)
```

**Description**

Solves a real symmetric positive or negative definite system \( Ax = b \) using a conjugate gradient method with or without preconditioning.

**Parameter**

- `b` – A `double` vector of length \( n \) containing the right-hand side.

**Returns**

A `double` vector of length \( n \) containing the approximate solution to the linear system.
Exceptions

System.ArgumentException is thrown if the length of b is not consistent with the order n of A.
Imsl.Math.SingularPreconditionMatrixException is thrown if the preconditioning matrix is singular.
Imsl.Math.NotDefinitePreconditionMatrixException is thrown if the preconditioning matrix is not definite.
Imsl.Math.SingularMatrixException is thrown if input matrix A is singular.
Imsl.Math.NotDefiniteAMatrixException is thrown if matrix A is not definite.
Imsl.Math.NoConvergenceException is thrown if the algorithm is not convergent within MaxIterations iterations.
Imsl.Math.NotDefiniteJacobiPreconditionerException is thrown if the Jacobi preconditioner is not definite.

Conjugate Gradient Example 1:
The solution to a positive definite linear system is found. The coefficient matrix is stored as a full matrix.

```csharp
using System;
using Imsl.Math;

public class ConjugateGradientEx1 : ConjugateGradient.IFunction
{
    private double[,] a = {
        {1.0, -3.0, 2.0},
        {-3.0, 10.0, -5.0},
        {2.0, -5.0, 6.0}};

    public void Amultp(double[] p, double[] z)
    {
        double[] w = Matrix.Multiply(a, p);
        Array.Copy(w, 0, z, 0, z.Length);
    }

    public static void Main(string[] args)
    {
        int n = 3;
        double[] b = {27.0, -78.0, 64.0};
        double[] solution = null;

        ConjugateGradientEx1 atp = new ConjugateGradientEx1();

        // Construct Cg object
        ConjugateGradient cg = new ConjugateGradient(n, atp);

        // Solve Ax=b
        solution = cg.Solve(b);
        new PrintMatrix("Solution").Print(solution);
    }
}
```
Output

Solution

0

0 1.00000000000116
1 -3.99999999999972
2 6.99999999999984

Conjugate Gradient Example 2:

In this example, two different preconditioners are used to find the solution of a sparse positive definite linear system which occurs in a finite difference solution of Laplace’s equation on a regular $c \times c$ grid, $c = 50$. The matrix is $A = E(c^2, c)$. For the first solution, Jacobi preconditioning with preconditioner $M = \text{diag}(A)$ is used. The required iteration number and maximum absolute error are printed. Next, a more complicated preconditioning matrix, consisting of the symmetric tridiagonal part of $A$, is used. Again, the iteration number and the maximum absolute error are printed. The iteration number is substantially reduced.

using System;
using Imsl.Math;

public class ConjugateGradientEx2 : ConjugateGradient.IPreconditioner
{
    private SparseMatrix A;
    private SparseCholesky M;

    public ConjugateGradientEx2(int n, int c)
    {
        // Create matrix $E(n,c)$, $n>1$, $1<c<n-1$
        // See Osterby and Zlatev(1982), pp. 7-8
        A = new SparseMatrix(n, n);
        for (int j = 0; j < n; j++)
        {
            if (j - c >= 0)
                A.Set(j, j - c, -1.0);
            if (j - 1 >= 0)
                A.Set(j, j - 1, -1.0);
            A.Set(j, j, 4.0);
            if (j + 1 < n)
                A.Set(j, j + 1, -1.0);
            if (j + c < n)
                A.Set(j, j + c, -1.0);
        }

        // Create and factor preconditioning matrix
        SparseMatrix C = new SparseMatrix(n, n);
        for (int j = 0; j < n; j++)
        {
            if (j - 1 >= 0)
                C.Set(j, j - 1, -1.0);
            C.Set(j, j, 4.0);
            if (j + 1 < n)
                C.Set(j, j + 1, -1.0);
        }
    }
}
C.Set(j, j + 1, -1.0);
}
M = new SparseCholesky(C);
M.FactorSymbolically();
M.FactorNumerically();
}

public void Amultp(double[] p, double[] z)
{
    double[] w = A.Multiply(p);
    Array.Copy(w, 0, z, 0, w.Length);
}

public void Preconditioner(double[] r, double[] z)
{
    double[] w = M.Solve(r);
    Array.Copy(w, 0, z, 0, w.Length);
}

public static void Main(String[] args)
{
    int n = 2500;
    int c = 50;

    ConjugateGradientEx2 atp = new ConjugateGradientEx2(n, c);

    // Set a predetermined answer and diagonal
    double[] expected = new double[n];
    double[] diagonal = new double[n];
    for (int i = 0; i < n; i++)
    {
        expected[i] = (double) (i % 5);
        diagonal[i] = 4.0;
    }

    // Get right-hand side
    double[] b = new double[n];
    atp.Amultp(expected, b);

    // Solve system with Jacobi preconditioning
    ConjugateGradient cgJacobi = new ConjugateGradient(n, atp);
    cgJacobi.SetJacobi(diagonal);
    double[] solution = cgJacobi.Solve(b);

    // Compute inf-norm of computed solution - exact solution, print results
    double norm = 0.0;
    for (int i = 0; i < n; i++)
    {
        norm = Math.Max(norm, Math.Abs(solution[i] - expected[i]));
    }
    Console.Out.WriteLine("Jacobi preconditioning");
    Console.Out.WriteLine("Iterations= " + cgJacobi.Iterations + ", norm= " + norm);
    Console.Out.WriteLine();
ConjugateGradient.IFunction Interface

public interface Imsl.Math.ConjugateGradient.IFunction
Public interface for the user supplied function to ConjugateGradient.

Method

Amultp
abstract public void Amultp(double[] p, double[] z)

Description
A user-supplied function which computes \( z = Ap \).

Parameters
- \( p \) – An input double vector of length dimension of \( A \).
- \( z \) – An output double vector containing the matrix-vector product \( Ap \).
ConjugateGradient.IPreconditioner Interface


Public interface for the user supplied function to ConjugateGradient used for preconditioning.

Method

Preconditioner
abstract public void Preconditioner(double[] r, double[] z)

Description
Used to compute \( z = M^{-1}r \) where \( M \) is the preconditioning matrix and \( r \) and \( z \) are arrays of length \( n \), the order of matrix \( M \).

Parameters

- \( r \) – An input double array of length \( n \) generated during the implementation of the Solve method.
- \( z \) – An output double array of length \( n \).
Chapter 2: Eigensystem Analysis

Types

class Eigen ................................................................. 132
class SymEigen .......................................................... 136

Usage Notes

An ordinary linear eigensystem problem is represented by the equation $Ax = \lambda x$ where $A$ denotes an $n \times n$ matrix. The value $\lambda$ is an eigenvalue and $x \neq 0$ is the corresponding eigenvector. The eigenvector is determined up to a scalar factor. In all functions, we have chosen this factor so that $x$ has Euclidean length one, and the component of $x$ of largest magnitude is positive. If $x$ is a complex vector, this component of largest magnitude is scaled to be real and positive. The entry where this component occurs can be arbitrary for eigenvectors having nonunique maximum magnitude values.

Error Analysis and Accuracy

Except in special cases, functions will not return the exact eigenvalue-eigenvector pair for the ordinary eigenvalue problem $Ax = \lambda x$. Typically, the computed pair

$$\tilde{x}, \tilde{\lambda}$$

is an exact eigenvector-eigenvalue pair for a “nearby” matrix $A + E$. Information about $E$ is known only in terms of bounds of the form $\|E\|_2 \leq f(n) \|A\|_2 \varepsilon$. The value of $f(n)$ depends on the algorithm, but is typically a small fractional power of $n$. The parameter $\varepsilon$ is the machine precision. By a theorem due to Bauer and Fike (see Golub and Van Loan 1989, p. 342),

$$\min \left| \tilde{\lambda} - \lambda \right| \leq \kappa(X) \|E\|_2 \text{ for all } \lambda \in \sigma(A)$$

where $\sigma(A)$ is the set of all eigenvalues of $A$ (called the spectrum of $A$), $X$ is the matrix of eigenvectors, $\|\cdot\|_2$ is Euclidean length, and $\kappa(X)$ is the condition number of $X$ defined as $\kappa(X) = \|X\|_2 \|X^{-1}\|_2$. If $A$ is a real symmetric or complex Hermitian matrix, then its eigenvector matrix $X$ is respectively orthogonal or unitary. For these matrices, $\kappa(X) = 1$.

The accuracy of the computed eigenvalues
and eigenvectors

\[ \tilde{\lambda}_j \]

\[ \tilde{x}_j \]

can be checked by computing their performance index \( \tau \). The performance index is defined to be

\[
\tau = \max_{1 \leq j \leq n} \frac{\| A \tilde{x}_j - \tilde{\lambda}_j \tilde{x}_j \|_2}{n \epsilon \| A \|_2 \| \tilde{x}_j \|_2}
\]

where \( \epsilon \) is again the machine precision.

The performance index \( \tau \) is related to the error analysis because

\[
\| E \tilde{x}_j \|_2 = \| A \tilde{x}_j - \tilde{\lambda}_j \tilde{x}_j \|_2
\]

where \( E \) is the “nearby” matrix discussed above.

While the exact value of \( \tau \) is precision and data dependent, the performance of an eigensystem analysis function is defined as excellent if \( \tau < 1 \), good if \( 1 \leq \tau \leq 100 \), and poor if \( \tau > 100 \). This is an arbitrary definition, but large values of \( \tau \) can serve as a warning that there is a significant error in the calculation.

If the condition number \( \kappa(X) \) of the eigenvector matrix \( X \) is large, there can be large errors in the eigenvalues even if \( \tau \) is small. In particular, it is often difficult to recognize near multiple eigenvalues or unstable mathematical problems from numerical results. This facet of the eigenvalue problem is often difficult for users to understand. Suppose the accuracy of an individual eigenvalue is desired. This can be answered approximately by computing the condition number of an individual eigenvalue (see Golub and Van Loan 1989, pp. 344-345). For matrices \( A \), such that the computed array of normalized eigenvectors \( X \) is invertible, the condition number of \( \lambda_i \) is

\[
\kappa_j = \| e_j^T X^{-1} \|,
\]

the Euclidean length of the \( j \)-th row of \( X^{-1} \). Users can choose to compute this matrix using the class LU in “Linear Systems.” An approximate bound for the accuracy of a computed eigenvalue is then given by \( \kappa_j \epsilon \| A \| \). To compute an approximate bound for the relative accuracy of an eigenvalue, divide this bound by \( |\tilde{\lambda}_j| \).

---

**Eigen Class**

```
public class Imsl.Math.Eigen

132 • Eigen  IMSL C# Numerical Library
```
Collection of Eigen System functions.

Eigen computes the eigenvalues and eigenvectors of a real matrix. The matrix is first balanced. Orthogonal similarity transformations are used to reduce the balanced matrix to a real upper Hessenberg matrix. The implicit double-shifted QR algorithm is used to compute the eigenvalues and eigenvectors of this Hessenberg matrix. The eigenvectors are normalized such that each has Euclidean length of value one. The largest component is real and positive.

The balancing routine is based on the EISPACK routine BALANC. The reduction routine is based on the EISPACK routines ORTHES and ORTRAN. The QR algorithm routine is based on the EISPACK routine HQR2. See Smith et al. (1976) for the EISPACK routines. Further details, some timing data, and credits are given in Hanson et al. (1990).

While the exact value of the performance index, $\tau$, is highly machine dependent, the performance of Eigen is considered excellent if $\tau < 1$, good if $1 \leq \tau \leq 100$, and poor if $\tau > 100$.

The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Smith et al. (1976, pages 124-125).

Properties

MaxIterations
virtual public int MaxIterations {get; set; }

Description
The maximum number of iterations.

Property Value
An int containing the maximum number of iterations.
Default: MaxIterations = 50.

Remarks
The maximum number of iterations must be greater than 0.

NumberOfProcessors
public int NumberOfProcessors {get; set; }

Description
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

Property Value
An int indicating the maximum possible number of processors to use.

Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount,
Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**Constructor**

**Eigen**

```csharp
public Eigen()
```

**Description**

Constructor for `Eigen`.

**Methods**

**GetValues**

```csharp
public Imsl.Math.Complex[] GetValues()
```

**Description**

Returns the eigenvalues of a matrix of type `double`.

**Returns**

A `Complex` array containing the eigenvalues of this matrix in descending order.

**GetVectors**

```csharp
public Imsl.Math.Complex[,] GetVectors()
```

**Description**

Returns the eigenvectors.

**Returns**

A `Complex` matrix containing the eigenvectors. The eigenvector corresponding to the j-th eigenvalue is stored in the j-th column. Each vector is normalized to have Euclidean length one.

**PerformanceIndex**

```csharp
public double PerformanceIndex(double[,] a)
```

**Description**

Returns the performance index of a real eigensystem.

**Parameter**

- `a` – A `double` matrix.

**Returns**

A `double` scalar value indicating how well the algorithms which have computed the eigenvalue and eigenvector pairs have performed.
Remarks
A performance index less than 1 is considered excellent, 1 to 100 is good, while greater than 100 is considered poor.

Solve
virtual public void Solve(double[,] a, bool computeVectors)

Description
Solves for the eigenvalues and (optionally) the eigenvectors of a real square matrix.

Parameters
a – A double square matrix for which the eigenvalues and (optionally) the eigenvectors are to be found.
computeVectors – A bool value of true if the eigenvectors are to be computed.

Exception
Imsl.Math.DidNotConvergeException is thrown when the algorithm fails to converge on the eigenvalues of the matrix

Example: Eigensystem Analysis
The eigenvalues and eigenvectors of a matrix are computed.

using System;
using Imsl.Math;

public class EigenEx1
{
    public static void Main(String[] args)
    {
        double[,] a = {
            {8, -1, -5},
            {-4, 4, -2},
            {18, -5, -7}
        };
        Eigen eigen = new Eigen();
        eigen.Solve(a, true);
        new PrintMatrix("Eigenvalues").SetPageWidth(80).Print(eigen.GetValues());
        new PrintMatrix("Eigenvectors").SetPageWidth(80).Print(eigen.GetVectors());
    }
}

Output

Eigenvalues

  0
  0  2+4i
  1  2-4i
  2 0.999999999999997

Eigensystem Analysis

Eigen • 135
SymEigen Class

public class Imsl.Math.SymEigen

Computes the eigenvalues and eigenvectors of a real symmetric matrix.

Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. These transformations are accumulated. An implicit rational QR algorithm is used to compute the eigenvalues of this tridiagonal matrix. The eigenvectors are computed using the eigenvalues as perfect shifts, Parlett (1980, pages 169, 172). The reduction routine is based on the EISPACK routine TRED2. See Smith et al. (1976) for the EISPACK routines. Further details, some timing data, and credits are given in Hanson et al. (1990).

Let \( M \) = the number of eigenvalues, \( \lambda \) = the array of eigenvalues, and \( x_j \) is the associated eigenvector with jth eigenvalue.

Also, let \( \varepsilon \) be the machine precision. The performance index, \( \tau \), is defined to be

\[
\tau = \max_{1 \leq j \leq M} \frac{\|Ax_j - \lambda_j x_j\|_1}{10N\varepsilon \|A\|_1 \|x_j\|_1}
\]

While the exact value of \( \tau \) is highly machine dependent, the performance of SymEigen is considered excellent if \( \tau < 1 \), good if \( 1 \leq 100 \), and poor if \( \tau > 100 \). The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Smith et al. (1976, pages 124-125).
Constructors

**SymEigen**

`public SymEigen(double[,] a)`

**Description**

Constructs the eigenvalues and the eigenvectors for a real symmetric matrix.

**Parameter**

- `a` – The symmetric matrix whose eigensystem is to be constructed.

**SymEigen**

`public SymEigen(double[,] a, bool computeVectors)`

**Description**

Constructs the eigenvalues and (optionally) the eigenvectors for a real symmetric matrix.

**Parameters**

- `a` – A `double` symmetric matrix whose eigensystem is to be constructed.
- `computeVectors` – A `bool`, `true` if the eigenvectors are to be computed.

Methods

**GetValues**

`public double[] GetValues()`

**Description**

Returns the eigenvalues.

**Returns**

A `double` array containing the eigenvalues in descending order.

**Remarks**

If the algorithm fails to converge on an eigenvalue, that eigenvalue is set to NaN.

**GetVectors**

`public double[,] GetVectors()`

**Description**

Return the eigenvectors of a symmetric matrix of type `double`.

**Returns**

A `double` array containing the eigenvectors.

Eigensystem Analysis
Remarks
The j-th column of the eigenvector matrix corresponds to the j-th eigenvalue. The eigenvectors are
normalized to have Euclidean length one. If the eigenvectors were not computed by the constructor, then
null is returned.

PerformanceIndex
public double PerformanceIndex(double[,] a)

Description
Returns the performance index of a real symmetric eigensystem.

Parameter
   a – A double symmetric matrix.

Returns
A double scalar value indicating how well the algorithms which have computed the eigenvalue and
eigenvector pairs have performed.

Remarks
A performance index less than 1 is considered excellent, 1 to 100 is good, while greater than 100 is
considered poor.

Example: Eigenvalues and Eigenvectors of a Symmetric Matrix
The eigenvalues and eigenvectors of a symmetric matrix are computed.

using System;
using Imsl.Math;

public class SymEigenEx1
{
   public static void Main(String[] args)
   {
      double[,] a = {
      {1, 1, 1},
      {1, 1, 1},
      {1, 1, 1}
      };

      SymEigen eigen = new SymEigen(a);
      new PrintMatrix("Eigenvalues").Print(eigen.GetValues());
      new PrintMatrix("Eigenvectors").Print(eigen.GetVectors());
   }
}

Output
   Eigenvalues
      0
      138
<table>
<thead>
<tr>
<th>0</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-3.53439879482695E-16</td>
</tr>
<tr>
<td>2</td>
<td>-2.22044604925031E-16</td>
</tr>
</tbody>
</table>

Eigenvectors

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.577350269189626</td>
<td>0.816496580927726</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.577350269189626</td>
<td>-0.408248290463863</td>
<td>-0.707106781186547</td>
</tr>
<tr>
<td>2</td>
<td>0.577350269189626</td>
<td>-0.408248290463863</td>
<td>0.707106781186548</td>
</tr>
</tbody>
</table>
Chapter 3: Interpolation and Approximation

Types

class Spline .......................................................... 143
class CsAkima ...................................................... 146
class CsTCB .......................................................... 163
class CsInterpolate ................................................ 152
enumeration CsInterpolate.Condition .......................... 154
class CsPeriodic ..................................................... 155
class CsShape .......................................................... 157
class CsSmooth ......................................................... 158
class CsSmoothC2 ..................................................... 160
class CsTCB .......................................................... 163
class BSpline .......................................................... 168
class BsInterpolate ................................................ 172
class BsLeastSquares ............................................... 174
class Spline2D .......................................................... 177
class Spline2DInterpolate .......................................... 180
class Spline2DLeastSquares ...................................... 189
class RadialBasis ..................................................... 195
interface RadialBasis.IFunction .................................. 207
class RadialBasis.Gaussian ........................................ 207
class RadialBasis.HardyMultiquadric ............................ 209

Usage Notes

This chapter contains classes to interpolate and approximate data with cubic splines. Interpolation means that the fitted curve passes through all of the specified data points. An approximation spline does not have to pass through any of the data points. An approximating curve can therefore be smoother than an interpolating curve.

Cubic splines are smooth $C^1$ or $C^2$ fourth-order piecewise-polynomial (pp) functions. For historical and
other reasons, cubic splines are the most heavily used pp functions.

This chapter contains four cubic spline interpolation classes and two approximation classes. These classes are derived from the base class Spline, which provides basic services, such as spline evaluation and integration.

The chart shows how the six cubic splines in this chapter fit a single data set.

Class CsInterpolate allows the user to specify various endpoint conditions (such as the value of the first and second derivatives at the right and left endpoints).

Class CsPeriodic is used to fit periodic (repeating) data. The sample data set used is not periodic and so the curve does not pass through the final data point.

Class CsAkima keeps the shape of the data while minimizing oscillations.

Class CsShape keeps the shape of the data by preserving its convexity.
Class CsSmooth constructs a smooth spline from noisy data.
Class CsSmoothC2 constructs a smooth spline from noisy data using cross-validation and a user-supplied smoothing parameter.

Class BSpline is the abstract base class for univariate B-splines. B-splines provide a particularly convenient and suitable basis for a given class of smooth piecewise polynomial (ppoly) functions. Such a class is specified by giving its breakpoint sequence, its order \( k \), and the required smoothness across each of the interior breakpoints. The corresponding B-spline basis is specified by giving its knot sequence \( t \in \mathbb{R} \). The specification rule is as follows: if the class is to have all derivatives up to and including the \( j \)-th derivative continuous across the interior breakpoint \( \xi_i \), then the number \( \xi_i \) should occur \( k-j-1 \) times in the knot sequence. Assuming that \( \xi_0 \) and \( \xi_{n-1} \) are the endpoints of the interval of interest, choose the first \( k \) knots equal to \( \xi_0 \) and the last \( k \) knots equal to \( \xi_{n-1} \). This can be done because the B-splines are defined to be right continuous near \( \xi_0 \) and left continuous near \( \xi_{n-1} \).

When the above construction is completed, a knot sequence \( t \) of length \( M \) is generated, and there are \( m = M-k \) B-splines of order \( k \), for example \( B_0, \ldots, B_{m-1} \), spanning the ppoly functions on the interval with the indicated smoothness. That is, each ppoly function in this class has a unique representation \( p = a_0B_0 + a_1B_1 + \cdots + a_{m-1}B_{m-1} \) as a linear combination of B-splines. A B-spline is a particularly compact ppoly function. \( B_i \) is a nonnegative function that is nonzero only on the interval \( [t_i, t_{i+k}] \). More precisely, the support of the \( i \)-th B-spline is \( [t_i, t_{i+k}] \). No ppoly function in the same class (other than the zero function) has smaller support (i.e., vanishes on more intervals) than a B-spline. This makes B-splines particularly attractive basis functions since the influence of any particular B-spline coefficient extends only over a few intervals. When it is necessary to emphasize the dependence of the B-spline on its parameters, we will use the notation \( B_{i,j,k} \) to denote the \( i \)-th B-spline of order \( k \) for the knot sequence \( t \).

Class BsInterpolate extends BSpline and creates a B-spline by interpolating data points.
Class BsLeastSquares extends BSpline and creates a B-spline by computing a least squares spline approximation to data points.

Class Spline2D is the abstract base class for the two-dimensional, tensor-product splines.

Class Spline2DInterpolate computes a Spline2D using interpolation from two-dimensional, tensor-product data.

Class RadialBasis computes an approximation to scattered data in \( \mathbb{R}^M \) using radial-basis functions.

---

**Spline Class**

```java
public class Imsl.Math.Spline

Spline represents and evaluates univariate piecewise polynomial splines.

A univariate piecewise polynomial (function) \( p(x) \) is specified by giving its breakpoint sequence \( \text{breakPoint[]} = [\xi] \in \mathbb{R}^n \), the order \( k \) (degree \( k-1 \)) of its polynomial pieces, and the \( k \times (n-1) \) matrix \( \text{coef=}c \) of its local polynomial coefficients. In terms of this information, the piecewise polynomial
```

---

**Interpolation and Approximation**

Spline • 143
(ppoly) function is given by

\[ p(x) = \sum_{j=1}^{k} c_{ji} \frac{(x - \xi_i)^{j-1}}{(j-1)!} \text{ for } \xi_i \leq x \leq \xi_{i+1} \]

The breakpoint sequence \( \xi_i \) is assumed to be strictly increasing, and we extend the ppoly function to the entire real axis by extrapolation from the first and last intervals.

**Constructor**

**Spline**

\`Spline()`

**Description**

Initializes a new instance of the Imsl.Math.Spline (p. 143) class.

**Methods**

**Derivative**

\`virtual public double Derivative(double x)`

**Description**

Returns the value of the first derivative of the spline at a point.

**Parameter**

\( x \) – A double, the point at which the derivative is to be evaluated.

**Returns**

A double containing the value of the first derivative of the spline at the point \( x \).

**Derivative**

\`virtual public double Derivative(double x, int ideriv)`

**Description**

Returns the value of the derivative of the spline at a point.

**Parameters**

\( x \) – A double, the point at which the derivative is to be evaluated.

\( ideriv \) – An int specifying the derivative to be computed. If zero, the function value is returned. If one, the first derivative is returned, etc.
**Returns**
A double containing the value of the derivative of the spline at the point \( x \).

**Derivative**

```csharp
virtual public double[] Derivative(double[] x, int ideriv)
```

**Description**
Returns the value of the derivative of the spline at each point of an array.

**Parameters**
- \( x \) – A double array of points at which the derivative is to be evaluated.
- \( ideriv \) – An int specifying the derivative to be computed. If zero, the function value is returned. If one, the first derivative is returned, etc.

**Returns**
A double array containing the value of the derivative the spline at each point of the array \( x \).

**Eval**

```csharp
virtual public double Eval(double x)
```

**Description**
Returns the value of the spline at a point.

**Parameter**
- \( x \) – A double, the point at which the spline is to be evaluated.

**Returns**
A double giving the value of the spline at the point \( x \).

**Eval**

```csharp
virtual public double[] Eval(double[] x)
```

**Description**
Returns the value of the spline at each point of an array.

**Parameter**
- \( x \) – A double array of points at which the spline is to be evaluated.

**Returns**
A double array containing the value of the spline at each point of the array \( x \).

**GetBreakpoints**

```csharp
public double[] GetBreakpoints()
```

**Description**
Returns a copy of the breakpoints.
Returns
A double array containing a copy of the breakpoints.

Integral
virtual public double Integral(double a, double b)

Description
Returns the value of an integral of the spline.

Parameters
a – A double specifying the lower limit of integration.
b – A double specifying the upper limit of integration.

Returns
A double, the integral of the spline from a to b.

CsAkima Class

public class Imsl.Math.CsAkima : Spline

Extension of the Spline class to handle the Akima cubic spline.

Class CsAkima computes a $C^1$ cubic spline interpolant to a set of data points $(x_i, f_i)$ for $i = 0, \ldots, n - 1$. The breakpoints of the spline are the abscissas. Endpoint conditions are automatically determined by the program; see Akima (1970) or de Boor (1978).

If the data points arise from the values of a smooth, say $C^4$, function $f$, i.e. $f_i = f(x_i)$, then the error will behave in a predictable fashion. Let $\xi$ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$
\| f - s \|_{[\xi_0, \xi_{n-1}]} \leq C \| f^{(2)} \|_{[\xi_0, \xi_{n-1}]} |\xi|^2
$$

where

$$
|\xi| := \max_{i=1, \ldots, n-1} |\xi_i - \xi_{i-1}|
$$

CsAkima is based on a method by Akima (1970) to combat wiggles in the interpolant. The method is nonlinear; and although the interpolant is a piecewise cubic, cubic polynomials are not reproduced. (However, linear polynomials are reproduced.)
Constructor

CsAkima
public CsAkima(double[] xData, double[] yData)

Description
Constructs the Akima cubic spline interpolant to the given data points.

Parameters
- **xData** – A double array containing the x-coordinates of the data. Values must be distinct.
- **yData** – A double array containing the y-coordinates of the data.

Exception
- `System.ArumentException` is thrown if the arrays `xData` and `yData` do not have the same length

Example: The Akima cubic spline interpolant

A cubic spline interpolant to a function is computed. The value of the spline at point 0.25 is printed.

```csharp
using System;
using Imsl.Math;

public class CsAkimaEx1
{
    public static void Main(String[] args)
    {
        int n = 11;
        double[] x = new double[n];
        double[] y = new double[n];

        for (int k = 0; k < n; k++)
        {
            x[k] = (double) k / (double) (n - 1);
            y[k] = Math.Sin(15.0 * x[k]);
        }

        CsAkima cs = new CsAkima(x, y);
        double csv = cs.Eval(0.25);
        Console.Out.WriteLine("The computed cubic spline value at point 0.25 is "+"csv");
    }
}
```

Output
The computed cubic spline value at point 0.25 is -0.478185519991867
public class Imsl.Math.CsTCB : Spline

Extension of the Spline class to handle a tension-continuity-bias (TCB) cubic spline, also known as a Kochanek-Bartels spline and is a generalization of the Catmull-Rom spline.

Let \( x = xData, y = yData, \) and \( n = \) the length of \( xData \) and \( yData. \) Class CsTCB computes the Kochanek-Bartels spline, a piecewise cubic Hermite spline interpolant to the set of data points \( \{x_i, y_i\} \) for \( i = 0, \ldots, n - 1. \) The breakpoints of the spline are the abscissas. As with all of the univariate interpolation functions, the abscissas need not be sorted.

The \( \{x_i\} \) values are the knots, so the \( i \)-th interval is \([x_i, x_{i+1}].\) (To simplify the explanation, it is assumed that the data points are given in increasing order.) The cubic Hermite in the \( i \)-th segment has a starting value of \( y_i \) and an ending value of \( y_{i+1}. \) Its incoming tangent is

\[
DS_i = \frac{1}{2}(1 - t_i)(1 - c_i)(1 + b_i) \frac{y_i - y_{i-1}}{x_{i+1} - x_i} + \frac{1}{2}(1 - t_i)(1 + c_i)(1 - b_i) \frac{y_{i+1} - y_i}{x_{i+1} - x_i}
\]

where \( t_i \) is the \( i \)-th tension value, \( c_i \) is the \( i \)-th continuity value, and \( b_i \) is the \( i \)-th bias value. Its outgoing tangent is

\[
DD_i = \frac{1}{2}(1 - t_i)(1 + c_i)(1 + b_i) \frac{y_i - y_{i-1}}{x_{i+1} - x_i} + \frac{1}{2}(1 - t_i)(1 - c_i)(1 - b_i) \frac{y_{i+1} - y_i}{x_{i+1} - x_i}
\]

The value of the tangent at the left endpoint is given as:

\[
y_0 - y_{-1} \\
x_1 - x_0
\]

The value of the tangent at the right endpoint is given as:

\[
y_n - y_{n-1} \\
x_n - x_{n-1}
\]

By default the values of the tangents at the leftmost and rightmost endpoints are zero. These values can be reset via the LeftEndTangent and RightEndTangent properties.

The spline has a continuous first derivative \((C^{-1})\) if at each data point the left and right tangents are equal. This is true if the continuity parameters, \( c_i, \) are all zero. For any values of the parameters the spline is continuous \((C^0).\)

If \( t_i = c_i = b_i = 0 \) for all \( i, \) then the curve is the Catmull-Rom spline.

The following chart shows the same data points interpolated with different parameter values. All of the tension, continuity, and bias parameters are zero except for the labeled parameter, which has the indicated value at all data points.
Tension controls how sharply the spline bends at the data points. The tension values can be set via the `SetTension` method. If tension values are near +1, the curve tightens. If the tension values are near -1, the curve slackens.

The continuity parameter controls the continuity of the first derivative. The continuity values can be set via the `SetContinuity` method. If the continuity value is zero, the spline’s first derivative is continuous, so the spline is $C^{-1}$.

The bias parameter controls the weighting of the left and right tangents. If zero, the tangents are equally weighted. If the bias parameter is near +1, the left tangent dominates. If the bias parameter is near -1, the right tangent dominates. The bias values can be set via the `SetBias` method.
Properties

**LeftEndTangent**

virtual public double LeftEndTangent {get; set; }

**Description**
The value of the tangent at the leftmost endpoint.

**Property Value**
A double containing the value of the tangent at the leftmost endpoint.
Default: LeftEndTangent is zero.

**RightEndTangent**

virtual public double RightEndTangent {get; set; }

**Description**
The value of the tangent at the rightmost endpoint.

**Property Value**
A double containing the value of the tangent at the rightmost endpoint.
Default: RightEndTangent is zero.

Constructor

**CsTCB**

class CsTCB
(public CsTCB(double[] xData, double[] yData)

**Description**
Constructs the tension-continuity-bias (TCB) cubic spline interpolant to the given data points.

**Parameters**
- xData – A double array containing the x-coordinates of the data.
- yData – A double array containing the y-coordinates of the data.

**Remarks**
Values must be distinct. xData and yData must be of the same length.

Methods

**Compute**

virtual public void Compute()
Computes the tension-continuity-bias (TCB) cubic spline interpolant.

SetBias
virtual public void SetBias(double[] bias)

Description
Sets the bias values at the data points.

Parameter
bias – A double array of length xData.Length which contains bias values in the interval [-1,1]. For each point, if the bias value is zero, the left and right side tangents are equally weighted. If the value is near +1, the left-side tangent dominates. If the value is near -1, the right side tangent dominates.

Default: All values of bias are zero.

SetContinuity
virtual public void SetContinuity(double[] continuity)

Description
Sets the continuity values at the data points.

Parameter
continuity – A double array of length xData.Length which contains continuity values in the interval [-1,1]. For each point, if the continuity value is zero, the curve is $C^1$ at that point. Otherwise, the curve has a corner at that point, but is still continuous ($C^0$).

Default: All values of continuity are zero.

SetTension
virtual public void SetTension(double[] tension)

Description
Sets the tension values at the data points.

Parameter
tension – A double array of length xData.Length which contains tension values in the interval [-1,1]. For each point, if the tension value is near +1, the curve is tightened at that point. If it is near -1, the curve is slack.

Default: All values of tension are zero.

Example: The Kochanek-Bartels cubic spline interpolant

This example interpolates to a set of points. At $x = 3$, the continuity and tension parameters are -1. At all other points, they are zero. Interpolated values are then printed.

```csharp
using System;
using Imsl.Math;
```

```csharp
using System;
using Imsl.Math;
```
```csharp
public class CsTCBEx1
{
    public static void Main(string[] args)
    {
        double[] xdata = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0};
        double[] ydata = {5.0, 2.0, 3.0, 5.0, 1.0, 2.0};
        double[] continuity = {0.0, 0.0, 0.0, -1.0, 0.0, 0.0};
        double[] tension = {0.0, 0.0, 0.0, -1.0, 0.0, 0.0};

        CsTCB cs = new CsTCB(xdata, ydata);
        cs.SetContinuity(continuity);
        cs.SetTension(tension);
        cs.Compute();
        Console.Out.WriteLine(" x " + " value ");
        for (int k = 0; k < 11; k++)
        {
            double x = (double) k / 2.0;
            double y = cs.Eval(x);
            Console.Out.WriteLine(" {0,5:0.0000} {1,5:0.0000}",x, y);
        }
    }
}
```

**Output**

<table>
<thead>
<tr>
<th>x</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>5.0000</td>
</tr>
<tr>
<td>0.5000</td>
<td>3.4375</td>
</tr>
<tr>
<td>1.0000</td>
<td>2.0000</td>
</tr>
<tr>
<td>1.5000</td>
<td>2.1875</td>
</tr>
<tr>
<td>2.0000</td>
<td>3.0000</td>
</tr>
<tr>
<td>2.5000</td>
<td>3.6875</td>
</tr>
<tr>
<td>3.0000</td>
<td>5.0000</td>
</tr>
<tr>
<td>3.5000</td>
<td>2.1875</td>
</tr>
<tr>
<td>4.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>4.5000</td>
<td>1.2500</td>
</tr>
<tr>
<td>5.0000</td>
<td>2.0000</td>
</tr>
</tbody>
</table>

### CsInterpolate Class

```csharp
public class Imsl.Math.CsInterpolate : Spline

Extension of the Spline class to interpolate data points.

CsInterpolate computes a $C^2$ cubic spline interpolant to a set of data points $(x_i, f_i)$ for $i = 0, \ldots, n - 1$. The breakpoints of the spline are the abscissas. Endpoint conditions are automatically determined by the program. These conditions correspond to the “not-a-knot” condition (see de Boor 1978), which requires that the third derivative of the spline be continuous at the second and next-to-last breakpoint. If $n$ is 2 or
```
3, then the linear or quadratic interpolating polynomial is computed, respectively.

If the data points arise from the values of a smooth, say, $C^4$ function $f$, i.e. $f_i = f(x_i)$, then the error will behave in a predictable fashion. Let $\xi$ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$|f - s|_{[\xi_0, \xi_n]} \leq C \|f^{(4)}\|_{[\xi_0, \xi_n]} |\xi|^4$$

where

$$|\xi| = \max_{i=0,...,n-1} |\xi_{i+1} - \xi_i|$$

For more details, see de Boor (1978, pages 55-56).

**Constructors**

**CsInterpolate**

```java
public CsInterpolate(double[] xData, double[] yData)
```

**Description**

Constructs a cubic spline that interpolates the given data points.

**Parameters**

- `xData` – A double array containing the x-coordinates of the data. Values must be distinct.
- `yData` – A double array containing the y-coordinates of the data. The arrays `xData` and `yData` must have the same length.

**CsInterpolate**

```java
public CsInterpolate(double[] xData, double[] yData,
                     Imsl.Math.CsInterpolate.Condition typeLeft, double valueLeft,
                     Imsl.Math.CsInterpolate.Condition typeRight, double valueRight)
```

**Description**

Constructs a cubic spline that interpolates the given data points with specified derivative endpoint conditions.

**Parameters**

- `xData` – A double array containing the x-coordinates of the data. Values must be distinct.
- `yData` – A double array containing the y-coordinates of the data. The arrays `xData` and `yData` must have the same length.
- `typeLeft` – A `CsInterpolate.Condition` denoting the type of condition at the left endpoint. This can be `NotAKnot`, `FirstDerivative` or `SecondDerivative`.
- `valueLeft` – A double value at the left endpoint. If `typeLeft` is `NotAKnot` this is ignored, otherwise, it is the value of the specified derivative.
A cubic spline interpolant to a function is computed. The value of the spline at point 0.25 is printed.

```csharp
class CsInterpolateEx1
{
    static void Main(string[] args)
    {
        int n = 11;
        double[] x = new double[n];
        double[] y = new double[n];

        for (int k = 0; k < n; k++)
        {
            x[k] = (double) k / (double) (n - 1);
            y[k] = Math.Sin(15.0 * x[k]);
        }

        CsInterpolate cs = new CsInterpolate(x, y);
        double csv = cs.Eval(0.25);
        Console.Out.WriteLine("The computed cubic spline value at 
                point .25 is "+csv);  
    }
}
```

Output

The computed cubic spline value at point .25 is -0.548772503812158

---

**CsInterpolate.Condition Enumeration**

```csharp
public enumeration Imsl.Math.CsInterpolate.Condition

Denotes the type of condition at an endpoint.
```
Fields

**FirstDerivative**

```java
public Imsl.Math.CsInterpolate.Condition FirstDerivative
```

**Description**
Satisfies the endpoint condition of the first derivative at the right and left points.

**NotAKnot**

```java
public Imsl.Math.CsInterpolate.Condition NotAKnot
```

**Description**
Satisfies the “not-a-knot” condition.

**SecondDerivative**

```java
```

**Description**
Satisfies the endpoint condition of the second derivative at the right and left points.

---

**CsPeriodic Class**

```java
public class Imsl.Math.CsPeriodic : Spline
```

Extension of the Spline class to interpolate data points with periodic boundary conditions.

Class CsPeriodic computes a $C^2$ cubic spline interpolant to a set of data points $(x_i, f_i)$ for $i = 0, \ldots, n - 1$. The breakpoints of the spline are the abscissas. The program enforces periodic endpoint conditions. This means that the spline $s$ satisfies $s(a) = s(b)$, $s'(a) = s'(b)$, and $s''(a) = s''(b)$, where $a$ is the leftmost abscissa and $b$ is the rightmost abscissa. If the ordinate values corresponding to $a$ and $b$ are not equal, then a warning message is issued. The ordinate value at $b$ is set equal to the ordinate value at $a$ and the interpolant is computed.

If the data points arise from the values of a smooth (say $C^4$) periodic function $f$, i.e. $f_i = f(x_i)$, then the error will behave in a predictable fashion. Let $\xi$ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$ |f - s|_{\xi_0, \xi_{n-1}} \leq C |f^{(4)}|_{\xi_0, \xi_{n-1}} |\xi|^4 $$

where

$$ |\xi| := \max_{i=1, \ldots, n-1} |\xi_i - \xi_{i-1}| $$

For more details, see de Boor (1978, pages 320-322).
**Constructor**

**CsPeriodic**

```java
public CsPeriodic(double[] xData, double[] yData)
```

**Description**
Constructs a cubic spline that interpolates the given data points with periodic boundary conditions.

**Parameters**
- `xData` – A `double` array containing the x-coordinates of the data. There must be at least 4 data points and values must be distinct.
- `yData` – A `double` array containing the y-coordinates of the data. The arrays `xData` and `yData` must have the same length.

**Example: The cubic spline interpolant with periodic boundary conditions**
A cubic spline interpolant to a function is computed. The value of the spline at point 0.23 is printed.

```java
using System;
using Imsl.Math;

public class CsPeriodicEx1
{
    public static void Main(String[] args)
    {
        int n = 11;
        double[] x = new double[n];
        double[] y = new double[n];

        double h = 2.0 * System.Math.PI / 15.0 / 10.0;
        for (int k = 0; k < n; k++)
        {
            x[k] = h * (double) (k);
            y[k] = System.Math.Sin(15.0 * x[k]);
        }

        CsPeriodic cs = new CsPeriodic(x, y);
        double csv = cs.Eval(0.23);
        Console.Out.WriteLine("The computed cubic spline value at ", +
                           "point .23 is " + csv);
    }
}
```

**Output**
The computed cubic spline value at point .23 is -0.303401472606451
CsShape Class

public class Imsl.Math.CsShape : Spline

Extension of the Spline class to interpolate data points consistent with the concavity of the data.

Class CsShape computes a cubic spline interpolant to \( n \) data points \( x_i, f_i \) for \( i = 0, \ldots, n - 1 \). For ease of explanation, we will assume that \( x_i < x_{i+1} \), although it is not necessary for the user to sort these data values. If the data are strictly convex, then the computed spline is convex, \( C^2 \), and minimizes the expression

\[
\int_{x_1}^{x_n} (g'')^2
\]

over all convex \( C^1 \) functions that interpolate the data. In the general case when the data have both convex and concave regions, the convexity of the spline is consistent with the data and the above integral is minimized under the appropriate constraints. For more information on this interpolation scheme, we refer the reader to Micchelli et al. (1985) and Irvine et al. (1986).

One important feature of the splines produced by this class is that it is not possible, a priori, to predict the number of breakpoints of the resulting interpolant. In most cases, there will be breakpoints at places other than data locations. The method is nonlinear; and although the interpolant is a piecewise cubic, cubic polynomials are not reproduced. However, linear polynomials are reproduced.) This routine should be used when it is important to preserve the convex and concave regions implied by the data.

Constructor

CsShape

public CsShape(double[] xData, double[] yData)

Description

Construct a cubic spline interpolant which is consistent with the concavity of the data.

Parameters

- \( xData \) – A double array containing the x-coordinates of the data. Values must be distinct.
- \( yData \) – A double array containing the y-coordinates of the data. The arrays \( xData \) and \( yData \) must have the same length.

Exceptions

- Imsl.Math.TooManyIterationsException is thrown if the iteration did not converge.
- Imsl.Math.SingularMatrixException is thrown if matrix is singular.
Example: The shape preserving cubic spline interpolant

A cubic spline interpolant to a function is computed consistent with the concavity of the data. The spline value at 0.05 is printed.

```csharp
using System;
using Imsl.Math;

public class CsShapeEx1
{
    public static void Main(String[] args)
    {
        double[] x = new double[]{0.00, 0.10, 0.20, 0.30, 0.40,
        0.50, 0.60, 0.80, 1.00};
        double[] y = new double[]{0.00, 0.90, 0.95, 0.90, 0.10,
        0.05, 0.05, 0.20, 1.00};

        CsShape cs = new CsShape(x, y);
        double csv = cs.Eval(0.05);
        Console.Out.WriteLine("The computed cubic spline value at " +
                            "point .05 is " + csv);
    }
}
```

Output

The computed cubic spline value at point .05 is 0.55823122286482

---

CsSmooth Class

```csharp
public class Imsl.Math.CsSmooth : Spline

Extension of the Spline class to construct a smooth cubic spline from noisy data points.

Class CsSmooth is designed to produce a \( C^2 \) cubic spline approximation to a data set in which the function values are noisy. This spline is called a smoothing spline. It is a natural cubic spline with knots at all the data abscissas \( x = xData \), but it does not interpolate the data \( (x_i, f_i) \). The smoothing spline \( S \) is the unique \( C^2 \) function that minimizes

\[
\int_a^b S''(x)^2 \, dx
\]

subject to the constraint
```
\[ \sum_{i=0}^{n-1} |(S(x_i) - f_i)w_i|^2 \leq \sigma \]

where \( \sigma \) is the smoothing parameter. The reader should consult Reinsch (1967) for more information concerning smoothing splines. CsSmooth solves the above problem when the user provides the smoothing parameter \( \sigma \). CsSmoothC2 attempts to find the “optimal” smoothing parameter using the statistical technique known as cross-validation. This means that (in a very rough sense) one chooses the value of \( \sigma \) so that the smoothing spline \( S_\sigma \) best approximates the value of the data at \( x_i \), if it is computed using all the data except the \( i \)-th; this is true for all \( i = 0, \ldots, n - 1 \). For more information on this topic, we refer the reader to Craven and Wahba (1979).

**Constructors**

**CsSmooth**

```java
public CsSmooth(double[] xData, double[] yData)
```

**Description**

Constructs a smooth cubic spline from noisy data using cross-validation to estimate the smoothing parameter. All of the points have equal weights.

**Parameters**

- `xData` – A `double` array containing the x-coordinates of the data. Values must be distinct.
- `yData` – A `double` array containing the y-coordinates of the data. The arrays `xData` and `yData` must have the same length.

**CsSmooth**

```java
public CsSmooth(double[] xData, double[] yData, double[] weight)
```

**Description**

Constructs a smooth cubic spline from noisy data using cross-validation to estimate the smoothing parameter. Weights are supplied by the user.

**Parameters**

- `xData` – A `double` array containing the x-coordinates of the data. Values must be distinct.
- `yData` – A `double` array containing the y-coordinates of the data. The arrays `xData` and `yData` must have the same length.
- `weight` – A `double` array containing the relative weights. This array must have the same length as `xData`.

**Example: The cubic spline interpolant to noisy data**

A cubic spline interpolant to noisy data is computed using cross-validation to estimate the smoothing parameter. The value of the spline at point 0.3010 is printed.
using System;
using Imsl.Math;
using Imsl.Stat;

public class CsSmoothEx1
{
    public static void Main(String[] args)
    {
        int n = 300;
        double[] x = new double[n];
        double[] y = new double[n];
        for (int k = 0; k < n; k++)
        {
            x[k] = (3.0 * k) / (n - 1);
            y[k] = 1.0 / (0.1 + System.Math.Pow(3.0 * (x[k] - 1.0), 4));
        }

        // Seed the random number generator
        rn.Multipier = 16807;

        // Contaminate the data
        for (int i = 0; i < n; i++)
        {
            y[i] += 2.0 * (float) rn.NextDouble() - 1.0;
        }

        // Smooth the data
        CsSmooth cs = new CsSmooth(x, y);
        double csv = cs.Eval(0.3010);
        Console.Out.WriteLine("The computed cubic spline value at " + ".3010 is " + csv);
    }
}

Output
The computed cubic spline value at point .3010 is 0.0101201298963992

CsSmoothC2 Class

public class Imsl.Math.CsSmoothC2 : Spline

Extension of the Spline class used to construct a spline for noisy data points using an alternate method.

Class CsSmoothC2 is designed to produce a $C^2$ cubic spline approximation to a data set in which the function values are noisy. This spline is called a smoothing spline. It is a natural cubic spline with knots at all the data abscissas $x$, but it does not interpolate the data $(x_i, f_i)$. The smoothing spline $S_\sigma$ is the
unique $C^2$ function that minimizes

$$\int_a^b s''_a(x)^2 \, dx$$

subject to the constraint

$$\sum_{i=0}^{n-1} |s_\sigma(x_i) - f_i|^2 \leq \sigma$$

Recommended values for $\sigma$ depend on the weights, $w$. If an estimate for the standard deviation of the error in the $y$-values is available, then $w_i$ should be set to this value and the smoothing parameter should be chosen in the confidence interval corresponding to the left side of the above inequality. That is,

$$n - \sqrt{2n} \leq \sigma \leq n + \sqrt{2n}$$

$CsSmoothC2$ is based on an algorithm of Reinsch (1967). This algorithm is also discussed in de Boor (1978, pages 235-243).

**Constructors**

$CsSmoothC2$

```java
public CsSmoothC2(double[] xData, double[] yData, double sigma)
```

**Description**

Constructs a smooth cubic spline from noisy data using an algorithm based on Reinsch (1967). All of the points have equal weights.

**Parameters**

- $xData$ – A double array containing the x-coordinates of the data. Values must be distinct.
- $yData$ – A double array containing the y-coordinates of the data. The arrays $xData$ and $yData$ must have the same length.
- $sigma$ – A double value specifying the smoothing parameter. $sigma$ must not be negative.

$CsSmoothC2$

```java
public CsSmoothC2(double[] xData, double[] yData, double[] weight, double sigma)
```

**Description**

Constructs a smooth cubic spline from noisy data using an algorithm based on Reinsch (1967) with weights supplied by the user.
Parameters

- xData – A double array containing the x-coordinates of the data. Values must be distinct.
- yData – A double array containing the y-coordinates of the data. The arrays xData and yData must have the same length.
- weight – A double array containing the weights. The arrays xData and weight must have the same length.
- sigma – A double value specifying the smoothing parameter. sigma must not be negative.

Example: The cubic spline interpolant to noisy data with supplied weights

A cubic spline interpolant to noisy data is computed using supplied weights and smoothing parameter. The value of the spline at point 0.3010 is printed.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;

public class CsSmoothC2Ex1
{
    public static void Main(String[] args)
    {
        // Set up a grid
        int n = 300;
        double[] x = new double[n];
        double[] y = new double[n];
        for (int k = 0; k < n; k++)
        {
            x[k] = 3.0 * ((double) (k) / (double) (n - 1));
            y[k] = 1.0 / (.1 + System.Math.Pow(3.0 * (x[k] - 1.0), 4));
        }

        // Seed the random number generator
        rn.Mulitplier = 16807;

        // Contaminate the data
        for (int i = 0; i < n; i++)
        {
            y[i] = y[i] + 2.0 * (float) rn.NextDouble() - 1.0;
        }

        // Set the weights
        double sdev = 1.0 / System.Math.Sqrt(3.0);
        double[] weights = new double[n];
        for (int i = 0; i < n; i++)
        {
            weights[i] = sdev;
        }

        // Set the smoothing parameter
    }
}
double smpar = (double) n;

// Smooth the data
CsSmoothC2 cs = new CsSmoothC2(x, y, weights, smpar);
double csv = cs.Eval(0.3010);
Console.Out.WriteLine("The computed cubic spline value at " +
    "point .3010 is " + csv);
}

Output
The computed cubic spline value at point .3010 is 0.0335028881575695

---

**CsTCB Class**

public class Imsl.Math.CsTCB : Spline

Extension of the Spline class to handle a tension-continuity-bias (TCB) cubic spline, also known as a Kochanek-Bartels spline and is a generalization of the Catmull-Rom spline.

Let \( x = x_{\text{Data}} \), \( y = y_{\text{Data}} \), and \( n \) = the length of \( x_{\text{Data}} \) and \( y_{\text{Data}} \). Class CsTCB computes the Kochanek-Bartels spline, a piecewise cubic Hermite spline interpolant to the set of data points \( \{x_i, y_i\} \) for \( i = 0, \ldots, n - 1 \). The breakpoints of the spline are the abscissas. As with all of the univariate interpolation functions, the abscissas need not be sorted.

The \( \{x_i\} \) values are the knots, so the \( i \)-th interval is \([x_i, x_{i+1}]\). (To simplify the explanation, it is assumed that the data points are given in increasing order.) The cubic Hermite in the \( i \)-th segment has a starting value of \( y_i \) and an ending value of \( y_{i+1} \). Its incoming tangent is

\[
DS_i = \frac{1}{2} (1 - t_i)(1 - c_i)(1 + b_i) \frac{y_i - y_{i-1}}{x_{i+1} - x_i} + \frac{1}{2} (1 - t_i)(1 + c_i)(1 - b_i) \frac{y_{i+1} - y_i}{x_{i+1} - x_i}
\]

where \( t_i \) is the \( i \)-th tension value, \( c_i \) is the \( i \)-th continuity value, and \( b_i \) is the \( i \)-th bias value. Its outgoing tangent is

\[
DD_i = \frac{1}{2} (1 - t_i)(1 + c_i)(1 + b_i) \frac{y_i - y_{i-1}}{x_{i+1} - x_i} + \frac{1}{2} (1 - t_i)(1 - c_i)(1 - b_i) \frac{y_{i+1} - y_i}{x_{i+1} - x_i}
\]

The value of the tangent at the left endpoint is given as:

\[
\frac{y_0 - y_{-1}}{x_1 - x_0}
\]

The value of the tangent at the right endpoint is given as:
By default the values of the tangents at the leftmost and rightmost endpoints are zero. These values can be reset via the `LeftEndTangent` and `RightEndTangent` properties.

The spline has a continuous first derivative ($C^{−1}$) if at each data point the left and right tangents are equal. This is true if the continuity parameters, $c_i$, are all zero. For any values of the parameters the spline is continuous ($C^0$).

If $t_i = c_i = b_i = 0$ for all $i$, then the curve is the Catmull-Rom spline.

The following chart shows the same data points interpolated with different parameter values. All of the tension, continuity, and bias parameters are zero except for the labeled parameter, which has the indicated value at all data points.

Tension controls how sharply the spline bends at the data points. The tension values can be set via the `SetTension` method. If tension values are near +1, the curve tightens. If the tension values are near -1, the curve slackens.

The continuity parameter controls the continuity of the first derivative. The continuity values can be set via the `SetContinuity` method. If the continuity value is zero, the spline’s first derivative is continuous, so the spline is $C^{−1}$.

The bias parameter controls the weighting of the left and right tangents. If zero, the tangents are equally weighted. If the bias parameter is near +1, the left tangent dominates. If the bias parameter is near -1, the right tangent dominates. The bias values can be set via the `SetBias` method.
Properties

**LeftEndTangent**

virtual public double LeftEndTangent {get; set; }

**Description**
The value of the tangent at the leftmost endpoint.

**Property Value**
A double containing the value of the tangent at the leftmost endpoint.
Default: LeftEndTangent is zero.

**RightEndTangent**

```csharp
virtual public double RightEndTangent {get; set; }
```

**Description**
The value of the tangent at the rightmost endpoint.

**Property Value**
A `double` containing the value of the tangent at the rightmost endpoint.

Default: RightEndTangent is zero.

---

**Constructor**

**CsTCB**

```csharp
public CsTCB(double[] xData, double[] yData)
```

**Description**
Constructs the tension-continuity-bias (TCB) cubic spline interpolant to the given data points.

**Parameters**
- `xData` – A `double` array containing the x-coordinates of the data.
- `yData` – A `double` array containing the y-coordinates of the data.

**Remarks**
Values must be distinct. `xData` and `yData` must be of the same length.

---

**Methods**

**Compute**

```csharp
virtual public void Compute()
```

**Description**
Computes the tension-continuity-bias (TCB) cubic spline interpolant.

**SetBias**

```csharp
virtual public void SetBias(double[] bias)
```

**Description**
Sets the bias values at the data points.
Parameter

*bias* – A double array of length `xData.Length` which contains bias values in the interval [-1,1]. For each point, if the bias value is zero, the left and right side tangents are equally weighted. If the value is near +1, the left-side tangent dominates. If the value is near -1, the right side tangent dominates.

Default: All values of `bias` are zero.

**SetContinuity**

```
virtual public void SetContinuity(double[] continuity)
```

**Description**

Sets the continuity values at the data points.

**Parameter**

*continuity* – A double array of length `xData.Length` which contains continuity values in the interval [-1,1]. For each point, if the continuity value is zero, the curve is $C^1$ at that point. Otherwise, the curve has a corner at that point, but is still continuous ($C^0$).  

Default: All values of `continuity` are zero.

**SetTension**

```
virtual public void SetTension(double[] tension)
```

**Description**

Sets the tension values at the data points.

**Parameter**

*tension* – A double array of length `xData.Length` which contains tension values in the interval [-1,1]. For each point, if the tension value is near +1, the curve is tightened at that point. If it is near -1, the curve is slack.  

Default: All values of `tension` are zero.

**Example: The Kochanek-Bartels cubic spline interpolant**

This example interpolates to a set of points. At x = 3, the continuity and tension parameters are -1. At all other points, they are zero. Interpolated values are then printed.

```csharp
using System;
using Imsl.Math;

class CsTCBE1
{
    public static void Main(string[] args)
    {
        double[] xdata = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0};
        double[] ydata = {5.0, 2.0, 3.0, 5.0, 1.0, 2.0};
        double[] continuity = {0.0, 0.0, 0.0, -1.0, 0.0, 0.0};
        double[] tension = {0.0, 0.0, 0.0, -1.0, 0.0, 0.0};
        CsTCB cs = new CsTCB(xdata, ydata);
    }
}
```
```csharp
cs.SetContinuity(continuity);
cs.SetTension(tension);
cs.Compute();
Console.Out.WriteLine("  x   \ value ");
for (int k = 0; k < 11; k++)
{
    double x = (double) k / 2.0;
    double y = cs.Eval(x);
    Console.Out.WriteLine(" {0,5:0.0000} {1,5:0.0000}", x, y);
}
```

Output

<table>
<thead>
<tr>
<th>x</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>5.0000</td>
</tr>
<tr>
<td>0.5000</td>
<td>3.4375</td>
</tr>
<tr>
<td>1.0000</td>
<td>2.0000</td>
</tr>
<tr>
<td>1.5000</td>
<td>2.1875</td>
</tr>
<tr>
<td>2.0000</td>
<td>3.0000</td>
</tr>
<tr>
<td>2.5000</td>
<td>3.6875</td>
</tr>
<tr>
<td>3.0000</td>
<td>5.0000</td>
</tr>
<tr>
<td>3.5000</td>
<td>2.1875</td>
</tr>
<tr>
<td>4.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>4.5000</td>
<td>1.2500</td>
</tr>
<tr>
<td>5.0000</td>
<td>2.0000</td>
</tr>
</tbody>
</table>

---

**BSpline Class**

```csharp
public class Imsl.Math.BSpline

Spline represents and evaluates univariate B-splines.

B-splines provide a particularly convenient and suitable basis for a given class of smooth ppoly functions. Such a class is specified by giving its breakpoint sequence, its order \(k\), and the required smoothness across each of the interior breakpoints. The corresponding B-spline basis is specified by giving its knot sequence \(t \in \mathbb{R}^M\). The specification rule is as follows: If the class is to have all derivatives up to and including the \(j\)-th derivative continuous across the interior breakpoint \(\xi_i\), then the number \(\xi_i\) should occur \(k-j\) times in the knot sequence. Assuming that \(\xi_1\) and \(\xi_n\) are the endpoints of the interval of interest, choose the first \(k\) knots equal to \(\xi_1\) and the last \(k\) knots equal to \(\xi_n\). This can be done because the B-splines are defined to be right continuous near \(\xi_1\) and left continuous near \(\xi_n\).

When the above construction is completed, a knot sequence \(t\) of length \(M\) is generated, and there are \(m = M-k\) B-splines of order \(k\), for example \(B_0, \ldots, B_{m-1}\), spanning the ppoly functions on the interval with the indicated smoothness. That is, each ppoly function in this class has a unique representation

\[
p = a_0B_0 + a_1B_1 + \ldots + a_{m-1}B_{m-1}
\]

as a linear combination of B-splines. A B-spline is a particularly
```
compact ppoly function. \( B_i \) is a nonnegative function that is nonzero only on the interval \([t_i, t_{i+k}]\). More precisely, the support of the \( i \)-th B-spline is \([t_i, t_{i+k}]\). No ppoly function in the same class (other than the zero function) has smaller support (i.e., vanishes on more intervals) than a B-spline. This makes B-splines particularly attractive basis functions since the influence of any particular B-spline coefficient extends only over a few intervals.

**Constructor**

**BSpline**

**BSpline()**

**Description**

Initializes a new instance of the Imsl.Math.BSpline (p. 168) class.

**Methods**

**Derivative**

**public double Derivative(double x)**

**Description**

Returns the value of the first derivative of the B-spline at a point.

**Parameter**

\( x \) – A double which specifies the point at which the derivative is to be evaluated.

**Returns**

A double containing the value of the first derivative of the B-spline at the point \( x \).

**Derivative**

**public double Derivative(double x, int ideriv)**

**Description**

Returns the value of the derivative of the B-spline at a point.

**Parameters**

\( x \) – A double which specifies the point at which the derivative is to be evaluated.

\( ideriv \) – A int specifying the derivative to be computed.

**Returns**

A double containing the value of the derivative of the B-spline at the point \( x \).
Remarks
If ideriv is zero, the function value is returned. If one, the first derivative is returned, etc.

**Derivative**
```
public double[] Derivative(double[] x, int ideriv)
```

**Description**
Returns the value of the derivative of the B-spline at each point of an array.

**Parameters**
- `x` – A `double` array of points at which the derivative is to be evaluated.
- `ideriv` – A `int` specifying the derivative to be computed.

**Returns**
A `double` array containing the value of the derivative the B-spline at each point of the array `x`.

Remarks
If `ideriv` is zero, the function value is returned. If one, the first derivative is returned, etc.

**Eval**
```
public double Eval(double x)
```

**Description**
Returns the value of the B-spline at a point.

**Parameter**
- `x` – A `double` which specifies the point at which the B-spline is to be evaluated.

**Returns**
A `double` giving the value of the B-spline at the point `x`.

**Eval**
```
public double[] Eval(double[] x)
```

**Description**
Returns the value of the B-spline at each point of an array.

**Parameter**
- `x` – A `double` array of points at which the B-spline is to be evaluated.

**Returns**
A `double` array containing the value of the B-spline at each point of the array `x`.

**GetKnots**
```
public double[] GetKnots()
```

**Description**
Returns a copy of the knot sequence.

IMSL C# Numerical Library
Returns
A double array containing a copy of the knot sequence.

GetSpline
public Imsl.Math.Spline GetSpline()

Description
Returns a Spline representation of the B-spline.

Returns
A Spline representation of the B-spline.

Integral
public double Integral(double a, double b)

Description
Returns the value of an integral of the B-spline.

Parameters
- a – A double specifying the lower limit of integration.
- b – A double specifying the upper limit of integration.

Returns
A double which specifies the integral of the B-spline from a to b.

Example: The B-spline interpolant
A B-Spline interpolant to data is computed. The value of the spline at point .23 is printed.

using System;
using Imsl.Math;

public class BsInterpolateEx1
{
    public static void Main(String[] args)
    {
        int n = 11;
        double[] x = new double[n];
        double[] y = new double[n];

        double h = 2.0 * System.Math.PI / 15.0 / 10.0;
        for (int k = 0; k < n; k++)
        {
            x[k] = h * (double) (k);
            y[k] = System.Math.Sin(15.0 * x[k]);
        }

        BsInterpolate bs = new BsInterpolate(x, y);
        double bsv = bs.Eval(0.23);
        Console.Out.WriteLine("The computed B-spline value at point ");
The computed B-spline value at point .23 is -0.303418399276769

Example: The B-spline least squares fit
A B-Spline least squares fit to data is computed. The value of the spline at point 4.5 is printed.

```csharp
using System;
using Imsl.Math;

public class BsLeastSquaresEx1
{
    public static void Main(String[] args)
    {
        double[] x = new double[]{0, 1, 2, 3, 4, 5, 8, 9, 10};
        double[] y = new double[]{1.0, 0.8, 2.4, 3.1, 4.5, 5.8, 6.2, 4.9, 3.7};
        BsLeastSquares bs = new BsLeastSquares(x, y, 5);
        double bsv = bs.Eval(4.5);
        Console.Out.WriteLine("The computed B-spline value at point " +
            ".23 is " + bsv);
    }
}
```

Output
The computed B-spline value at point 4.5 is 5.22855432359694

BsInterpolate Class

```csharp
public class Imsl.Math.BsInterpolate : BSpline
Extension of the BSpline class to interpolate data points.

Given the data points x = xData, f = yData, and n the number of elements in xData and yData, the
default action of BsInterpolate computes a cubic (order = 4) spline interpolant s to the data using a
default “not-a-knot” knot sequence. Constructors are also provided that allow the order and knot
sequence to be specified. This algorithm is based on the routine SPLINT by de Boor (1978, p. 204).

First, the xData vector is sorted and the result is stored in x. The elements of yData are permuted
appropriately and stored in f, yielding the equivalent data (x_i, f_i) for i = 0 to n-1. The following
preliminary checks are performed on the data, with k = order. We verify that
```
\[ x_i < x_{i+1} \text{ for } i = 0, \ldots, n - 2 \]
\[ t_i < t_{i+k} \text{ for } i = 0, \ldots, n - 1 \]
\[ t_i < t_{i+1} \text{ for } i = 0, \ldots, n+k - 2 \]

The first test checks to see that the abscissas are distinct. The second and third inequalities verify that a valid knot sequence has been specified.

In order for the interpolation matrix to be nonsingular, we also check \( t_{k-1} \leq x_i \leq t_n \) for \( i = 0 \) to \( n - 1 \). This first inequality in the last check is necessary since the method used to generate the entries of the interpolation matrix requires that the \( k \) possibly nonzero B-splines at \( x_i, B_{j-k+1}, \ldots, B_j \) where \( j \) satisfies \( t_j \leq x_i \) be well-defined (that is, \( j - k + 1 \geq 0 \)).

### Constructors

**BsInterpolate**

```java
public BsInterpolate(double[] xData, double[] yData)
```

**Description**

Constructs a B-spline that interpolates the given data points. The computed B-spline will be order 4 (cubic) and have a default “not-a-knot” spline knot sequence.

**Parameters**

- `xData` – A `double` array containing the x-coordinates of the data. Values must be distinct.
- `yData` – A `double` array containing the y-coordinates of the data. The arrays `xData` and `yData` must have the same length.

**BsInterpolate**

```java
public BsInterpolate(double[] xData, double[] yData, int order)
```

**Description**

Constructs a B-spline that interpolates the given data points and order, using a default “not-a-knot” spline knot sequence.

**Parameters**

- `xData` – A `double` array containing the x-coordinates of the data. Values must be distinct.
- `yData` – A `double` array containing the y-coordinates of the data. The arrays `xData` and `yData` must have the same length.
- `order` – An `int` denoting the order of the B-spline.

**BsInterpolate**

```java
public BsInterpolate(double[] xData, double[] yData, int order, double[] knot)
```

**Description**

Constructs a B-spline that interpolates the given data points, using the specified order and knots.

---

Interpolation and Approximation • BsInterpolate • 173
Parameters

- xData – A double array containing the x-coordinates of the data. Values must be distinct.
- yData – A double array containing the y-coordinates of the data. The arrays xData and yData must have the same length.
- order – A int denoting the order of the spline.
- knot – A double array containing the knot sequence for the B-spline.

Example: The B-spline interpolant

A B-Spline interpolant to data is computed. The value of the spline at point 0.23 is printed.

```csharp
using System;
using Imsl.Math;

public class BsInterpolateEx1
{
    public static void Main(String[] args)
    {
        int n = 11;
        double[] x = new double[n];
        double[] y = new double[n];

        double h = 2.0 * System.Math.PI / 15.0 / 10.0;
        for (int k = 0; k < n; k++)
        {
            x[k] = h * (double) (k);
            y[k] = System.Math.Sin(15.0 * x[k]);
        }

        BsInterpolate bs = new BsInterpolate(x, y);
        double bsv = bs.Eval(0.23);
        Console.Out.WriteLine("The computed B-spline value at point " + ".23 is " + bsv);
    }
}
```

Output

The computed B-spline value at point .23 is -0.303418399276769

BsLeastSquares Class

public class Imsl.Math.BsLeastSquares : BSpline

Extension of the BSpline class to compute a least squares spline approximation to data points.
Let's make the identifications
\[ n = xData.Length \]
\[ x = xData \]
\[ f = yData \]
\[ m = nCoef \]
\[ k = \text{order} \]

For convenience, we assume that the sequence \( x \) is increasing, although the class does not require this.

By default, \( k = 4 \), and the knot sequence we select equally distributes the knots through the distinct \( x_i \)’s. In particular, the \( m + k \) knots will be generated in \([x_1, x_n]\) with \( k \) knots stacked at each of the extreme values. The interior knots will be equally spaced in the interval.

Once knots \( t \) and weights \( w \) are determined, then the spline least-squares fit to the data is computed by minimizing over the linear coefficients \( a_j \)

\[
\sum_{i=0}^{n-1} w_i \left[ f_i - \sum_{j=1}^{m} a_j B_j(x_i) \right] ^2
\]

where the \( B_j, j = 1, \ldots, m \) are a (B-spline) basis for the spline subspace.

This algorithm is based on the routine L2APPR by deBoor (1978, p. 255).

**Constructors**

**BsLeastSquares**

```java
public BsLeastSquares(double[] xData, double[] yData, int nCoef)
```

**Description**

Constructs a least squares B-spline approximation to the given data points.

**Parameters**

- \( xData \) – A double array containing the x-coordinates of the data.
- \( yData \) – A double array containing the y-coordinates of the data. The arrays \( xData \) and \( yData \) must have the same length.
- \( nCoef \) – A int denoting the linear dimension of the spline subspace. It should be smaller than the number of data points and greater than or equal to the order of the spline (whose default value is 4).

**BsLeastSquares**

```java
public BsLeastSquares(double[] xData, double[] yData, int nCoef, int order)
```

**Description**

Constructs a least squares B-spline approximation to the given data points.
Parameters

- **xData** – A double array containing the x-coordinates of the data.
- **yData** – A double array containing the y-coordinates of the data. The arrays xData and yData must have the same length.
- **nCoef** – A int denoting the linear dimension of the spline subspace. It should be smaller than the number of data points and greater than or equal to the order of the spline.
- **order** – A int denoting the order of the spline.

BsLeastSquares

custom.BsLeastSquares(double[] xData, double[] yData, int nCoef, int order, double[] weight, double[] knot)

Description

Constructs a least squares B-spline approximation to the given data points.

Parameters

- **xData** – A double array containing the x-coordinates of the data.
- **yData** – A double array containing the y-coordinates of the data. The arrays xData and yData must have the same length.
- **nCoef** – A int denoting the linear dimension of the spline subspace. It should be smaller than the number of data points and greater than or equal to the order of the spline.
- **order** – A int denoting the order of the spline.
- **weight** – A double array containing the weights for the data. The arrays xData, yData and weight must have the same length.
- **knot** – A double array containing the knot sequence for the spline.

Example: The B-spline least squares fit

A B-Spline least squares fit to data is computed. The value of the spline at point 4.5 is printed.

```csharp
using System;
using Imsl.Math;

public class BsLeastSquaresEx1
{
    public static void Main(String[] args)
    {
        double[] x = new double[]{0, 1, 2, 3, 4, 5, 8, 9, 10};
        double[] y = new double[]{1.0, 0.8, 2.4, 3.1, 4.5, 5.8, 6.2, 4.9, 3.7};

        BsLeastSquares bs = new BsLeastSquares(x, y, 5);
        double bsv = bs.Eval(4.5);
        Console.Out.WriteLine("The computed B-spline value at point " +
                                "4.5 is " + bsv);
    }
}
```
Output
The computed B-spline value at point 4.5 is 5.22855432359694

Spline2D Class

public class Imsl.Math.Spline2D
Represents and evaluates tensor-product splines.

The simplest method of obtaining multivariate interpolation and approximation functions is to take univariate methods and form a multivariate method via tensor products. In the case of two-dimensional spline interpolation, the derivation proceeds as follows: Let $t_x$ be a knot sequence for splines of order $k_x$, and $t_y$ be a knot sequence for splines of order $k_y$. Let $N_x + k_x$ be the length of $t_x$, and $N_y + k_y$ be the length of $t_y$. Then, the tensor-product spline has the following form:

$$
\sum_{m=0}^{N_y-1} \sum_{n=0}^{N_x-1} c_{nm} B_{m,k_x,t_x}(x) B_{n,k_y,t_y}(y)
$$

Given two sets of points

$$\{x_i\}_{i=1}^{N_x}$$

and

$$\{y_j\}_{j=1}^{N_y}$$

for which the corresponding univariate interpolation problem can be solved, the tensor-product interpolation problem finds the coefficients $c_{nm}$ so that

$$
\sum_{m=0}^{N_y-1} \sum_{n=0}^{N_x-1} c_{nm} B_{m,k_x,t_x}(x_i) B_{n,k_y,t_y}(y_j) = f_{ij}
$$

This problem can be solved efficiently by repeatedly solving univariate interpolation problems as described in de Boor (1978, p. 347).

Constructor

Spline2D
Spline2D()

Description
Initializes a new instance of the Imsl.Math.Spline2D (p. 177) class.
Methods

Derivative
virtual public double Derivative(double x, double y, int xPartial, int yPartial)

Description
Returns the value of the partial derivative of the tensor-product spline at the point \((x, y)\).

Parameters
- \(x\) – A double scalar specifying the \(x\)-coordinate of the evaluation point for the tensor-product spline.
- \(y\) – A double scalar specifying the \(y\)-coordinate of the evaluation point for the tensor-product spline.
- \(xPartial\) – An int scalar specifying the \(x\)-partial derivative.
- \(yPartial\) – An int scalar specifying the \(y\)-partial derivative.

Returns
A double scalar containing the value of the partial derivative
\[
\frac{\partial^i \partial^j s}{\partial x \partial y}
\]
where \(i = xPartial\) and \(j = yPartial\), at \((x, y)\).

Derivative
virtual public double[,] Derivative(double[] xVec, double[] yVec, int xPartial, int yPartial)

Description
Returns the values of the partial derivative of the tensor-product spline of an array of points.

Parameters
- \(xVec\) – A double array specifying the \(x\)-coordinates at which the spline is to be evaluated.
- \(yVec\) – A double array specifying the \(y\)-coordinates at which the spline is to be evaluated.
- \(xPartial\) – An int scalar specifying the \(x\)-partial derivative.
- \(yPartial\) – An int scalar specifying the \(y\)-partial derivative.

Returns
An double matrix containing the values of the partial derivatives
\[
\frac{\partial^i \partial^j s}{\partial x \partial y}
\]
where \(i = xPartial\) and \(j = yPartial\), at each \((x, y)\).

GetCoefficients
virtual public double[,] GetCoefficients()
Returns the coefficients for the tensor-product spline.

**GetXKnots**
virtual public double[] GetXKnots()

Returns the knot sequences in the x-direction.

**GetYKnots**
virtual public double[] GetYKnots()

Returns the knot sequences in the y-direction.

**Integral**
public double Integral(double a, double b, double c, double d)

Returns the value of an integral of a tensor-product spline on a rectangular domain.

**Parameters**
- a – A double specifying the lower limit for the first variable of the tensor-product spline.
- b – A double specifying the upper limit for the first variable of the tensor-product spline.
- c – A double specifying the lower limit for the second variable of the tensor-product spline.
- d – A double specifying the upper limit for the second variable of the tensor-product spline.

**Returns**
A double, the integral of the tensor-product spline over the rectangle [a, b] by [c, d].

**Remarks**
If s is the spline, then the Integral method returns

\[ \int_a^b \int_c^d s(x,y)dydx \]

This method uses the (univariate integration) identity (22) in de Boor (1978, p. 151)

\[ \int_0^{r-1} \sum_{i=0}^{\tau_n-1} \alpha_i B_{i,k} (\tau) \, d\tau = \sum_{i=0}^{r-1} \left[ \sum_{j=0}^{i} \alpha_j \frac{t_{j+k} - t_j}{k} \right] B_{i,k+1} (x) \]
where $t_0 \leq x \leq t_r$.

It assumes (for all knot sequences) that the first and last $k$ knots are stacked, that is, $t_0 = \ldots = t_{k-1}$ and $t_n = \ldots = t_{n+k-1}$, where $k$ is the order of the spline in the $x$ or $y$ direction.

**Value**

virtual public double Value(double x, double y)

**Description**

Returns the value of the tensor-product spline at the point $(x, y)$.

**Parameters**

- $x$ – A double scalar specifying the $x$-coordinate of the evaluation point for the tensor-product spline.
- $y$ – A double scalar specifying the $y$-coordinate of the evaluation point for the tensor-product spline.

**Returns**

A double scalar containing the value of the tensor-product spline.

**Value**

virtual public double[,] Value(double[] xVec, double[] yVec)

**Description**

Returns the values of the tensor-product spline of an array of points.

**Parameters**

- $xVec$ – A double array specifying the $x$-coordinates at which the spline is to be evaluated.
- $yVec$ – A double array specifying the $y$-coordinates at which the spline is to be evaluated.

**Returns**

A double matrix containing the values evaluated.

---

**Spline2DInterpolate Class**

public class Imsl.Math.Spline2DInterpolate : Spline2D

Computes a two-dimensional, tensor-product spline interpolant from two-dimensional, tensor-product data.

The class Spline2DInterpolate computes a tensor-product spline interpolant. The tensor-product spline interpolant to data $\{(x_i, y_j, f_{ij})\}$, where $0 \leq i \leq (n_x-1)$ and $0 \leq j \leq (n_y-1)$ has the form

$$
\sum_{m=0}^{n_y-1} \sum_{n=0}^{n_x-1} c_{nm} B_{n,k_x,t_x}(x) B_{m,k_y,t_y}(y)
$$

---
where \( k_x \) and \( k_y \) are the orders of the splines. These numbers are defaulted to be 4, but can be set to any positive integer using `xOrder` and `yOrder` in the constructor. Likewise, \( t_x \) and \( t_y \) are the corresponding knot sequences (`xKnots` and `yKnots`). These default values are selected by `Spline2DInterpolate`. The algorithm requires that

\[
t_x(k_x - 1) \leq x_i \leq t_x(n_x) \quad 0 \leq i \leq n_x - 1
\]

\[
t_y(k_y - 1) \leq y_j \leq t_y(n_y - 1) \quad 0 \leq j \leq n_y - 1
\]

Tensor-product spline interpolants in two dimensions can be computed quite efficiently by solving (repeatedly) two univariate interpolation problems.

The computation is motivated by the following observations. It is necessary to solve the system of equations

\[
\sum_{m=0}^{n_x-1} \sum_{n=0}^{n_y-1} c_{nm} B_{n,k_x,t_x}(x_i) B_{m,k_y,t_y}(y_j) = f_{ij}
\]

Setting

\[
h_{mi} = \sum_{n=0}^{n_x-1} c_{nm} B_{n,k_x,t_x}(x_i)
\]

note that for each fixed \( i \) from 0 to \( n_x - 1 \), we have \( n_y \) linear equations in the same number of unknowns as can be seen below:

\[
\sum_{m=0}^{n_y-1} h_{mi} B_{m,k_y,t_y}(y_j) = f_{ij}
\]

\[
\sum_{m=0}^{n_x-1} \sum_{n=0}^{n_y-1} c_{nm} B_{n,k_x,t_x}(x_i) B_{m,k_y,t_y}(y_j) = f_{ij}
\]

Setting

\[
h_{mi} = \sum_{n=0}^{n_y-1} c_{nm} B_{n,k_y,t_y}(y_j)
\]

note that for each fixed \( i \) from 0 to \( n_x - 1 \), we have \( n_y - 1 \) linear equations in the same number of unknowns as can be seen below:

\[
\sum_{m=0}^{n_y-1} h_{mi} B_{m,k_y,t_y}(y_j) = f_{ij}
\]

The same matrix appears in all of the equations above:

\[
[B_{m,k_y,t_y}(y_j)] \quad 0 \leq m, j \leq n_y - 1
\]

Thus, only factor this matrix once and then apply this factorization to the \( n_x \) right-hand sides. Once this is done and \( h_{mi} \) is computed, then solve for the coefficients \( c_{nm} \) using the relation

\[
\sum_{n=0}^{n_x-1} c_{nm} B_{n,k_x,t_x}(x_i) = h_{mi}
\]

for \( n \) from 0 to \( n_y - 1 \), which again involves one factorization and \( n_y \) solutions to the different right-hand sides. The class `Spline2DInterpolate` is based on the routine SPLI2D by de Boor (1978, p. 347).
Constructors

Spline2DInterpolate
public Spline2DInterpolate(double[] xData, double[] yData, double[,] fData)

Description
Constructor for Spline2DInterpolate.

Parameters
- xData – A double array containing the data points in the x-direction.
- yData – A double array containing the data points in the y-direction.
- fData – A double matrix of size xData.Length by yData.Length containing the values to be interpolated.

Spline2DInterpolate
public Spline2DInterpolate(double[] xData, double[] yData, double[,] fData, int xOrder, int yOrder)

Description
Constructor for Spline2DInterpolate.

Parameters
- xData – A double array containing the data points in the x-direction.
- yData – A double array containing the data points in the y-direction.
- fData – A double matrix of size xData.Length by yData.Length containing the values to be interpolated.
- xOrder – An int scalar value specifying the order of the spline in the x-direction. xOrder must be at least 1. By default, xOrder = 4, tensor-product cubic spline.
- yOrder – An int scalar value specifying the order of the spline in the y-direction. yOrder must be at least 1. By default, yOrder = 4, tensor-product cubic spline.

Spline2DInterpolate
public Spline2DInterpolate(double[] xData, double[] yData, double[,] fData, int xOrder, int yOrder, double[] xKnots, double[] yKnots)

Description
Constructor for Spline2DInterpolate.

Parameters
- xData – A double array containing the data points in the x-direction.
- yData – A double array containing the data points in the y-direction.
- fData – A double matrix of size xData.Length by yData.Length containing the values to be interpolated.
xOrder – An int scalar value specifying the order of the spline in the x-direction. xOrder must be at least 1. By default, xOrder = 4, tensor-product cubic spline.

yOrder – An int scalar value specifying the order of the spline in the y-direction. yOrder must be at least 1. By default, yOrder = 4, tensor-product cubic spline.

xKnots – A double array of size xData.Length + xOrder specifying the knot sequences of the spline in the x-direction. By default, knot sequences are selected by the class.

yKnots – A double array of size yData.Length + yOrder specifying the knot sequences of the spline in the y-direction. By default, knot sequences are selected by the class.

**Example 1**

A tensor-product spline interpolant to a function

\[ f(x, y) = x^3 + y^2 \]

is computed. The values of the interpolant and the error on a 4 x 4 grid are displayed.

```csharp
using System;
using Imsl.Math;

class Spline2DInterpolateEx1
{
    private static double F(double x, double y)
    {
        return (x * x * x + y * y);
    }

    public static void Main(string[] args)
    {
        int nData = 11;
        int outData = 2;
        double[,] fdata = new double[nData, nData];
        double[] xData = new double[nData];
        double[] yData = new double[nData];
        double x, y, z;

        // Set up grid
        for (int i = 0; i < nData; i++)
        {
            xData[i] = yData[i] = (double) i / ((double) (nData - 1));
        }

        for (int i = 0; i < nData; i++)
        {
            for (int j = 0; j < nData; j++)
            {
                fdata[i, j] = F(xData[i], yData[j]);
            }
        }

        // Compute tensor-product interpolant
    }
}
```
Spline2DInterpolate spline = new Spline2DInterpolate(xData, yData, fdata);

// Print results
Console.Out.WriteLine(" x y F(x, y) "
+ "Interpolant Error");
for (int i = 0; i < outData; i++)
{
    x = (double) i / (double) (outData);
    for (int j = 0; j < outData; j++)
    {
        y = (double) j / (double) (outData);
        z = spline.Value(x, y);
        Console.Out.WriteLine(x.ToString("0.0000") + "
" + y.ToString("0.0000") + " +F(x,y).ToString("0.0000")
+"+z.ToString("0.0000")" + "
+Math.Abs(F(x, y) - z).ToString("0.0000");
    }
}

Output

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F(x, y)</th>
<th>Interpolant</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.5000</td>
<td>0.2500</td>
<td>0.2500</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.5000</td>
<td>0.0000</td>
<td>0.1250</td>
<td>0.1250</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.5000</td>
<td>0.5000</td>
<td>0.3750</td>
<td>0.3750</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Example 2

A tensor-product spline interpolant to a function

\[ f(x, y) = x^3 + y^2 \]

is computed. The values of the interpolant and the error on a 4 x 4 grid are displayed. Notice that the first interpolant with order = 3 does not reproduce the cubic data, while the second interpolant with order = 6 does reproduce the data.

using System;
using Imsl.Math;

public class Spline2DInterpolateEx2
{
    private static double F(double x, double y)
    {
        return (x * x * x + y * y);
    }

    public static void Main(String[] args)
    {
int nData = 7;
int outData = 4;
double[,] fData = new double[nData,nData];
double[] xData = new double[nData];
double[] yData = new double[nData];
double x, y, z;

// Set up grid
for (int i = 0; i < nData; i++)
{
    xData[i] = yData[i] = (double) i / ((double) (nData - 1));
}

for (int i = 0; i < nData; i++)
{
    for (int j = 0; j < nData; j++)
    {
        fData[i,j] = F(xData[i], yData[j]);
    }
}

for (int order = 3; order < 7; order += 3)
{
    // Compute tensor-product interpolant
    Spline2DInterpolate spline = new Spline2DInterpolate(xData, yData, 
        fData, order, order);
    // Print results
    Console.Out.WriteLine("\nThe order of the spline is "+ order);
    Console.Out.WriteLine(" x y F(x, y) " + "Interpolant Error");
    for (int i = 0; i < outData; i++)
    {
        x = (double) i / (double) (outData);
        for (int j = 0; j < outData; j++)
        {
            y = (double) j / (double) (outData);
            z = spline.Value(x, y);
            Console.Out.WriteLine(x.ToString("0.0000") +" " + y.ToString("0.0000") +" +F(x,y).ToString("0.0000") +" +Math.Abs(F(x, y) - z).ToString("0.0000"));
        }
    }
}

**Output**
The order of the spline is 3
The order of the spline is 6

Example 3

A spline interpolant \( s \) to a function

\[
f(x,y) = x^3 y^2
\]

is constructed. Then, the values of the partial derivative

\[
\frac{\partial^3 s(x,y)}{\partial x^2 \partial y}
\]

and the error are computed on a 4 x 4 grid.

```csharp
using System;
using Imsl.Math;

public class Spline2DInterpolateEx3
{

186 ● Spline2DInterpolate IMSL C# Numerical Library
```
private static double F(double x, double y) {
    return (x * x * x * y * y);
}

private static double F21(double x, double y) {
    return (6.0 * x * 2.0 * y);
}

public static void Main(String[] args) {
    int nData = 11;
    int outData = 2;
    double[,] fData = new double[nData, nData];
    double[] xData = new double[nData];
    double[] yData = new double[nData];
    double x, y, z;

    // Set up grid
    for (int i = 0; i < nData; i++)
    {
        xData[i] = yData[i] = (double) i / ((double) (nData - 1));
    }

    for (int i = 0; i < nData; i++)
    {
        for (int j = 0; j < nData; j++)
        {
            fData[i, j] = F(xData[i], yData[j]);
        }
    }

    // Compute tensor-product interpolant
    Spline2DInterpolate spline = new Spline2DInterpolate(xData, yData, fData);

    // Print results
    Console.Out.WriteLine(" x y F21(x, y) 21InterpDeriv Error");
    for (int i = 0; i < outData; i++)
    {
        x = (double) (1 + i) / (double) (outData + 1);
        for (int j = 0; j < outData; j++)
        {
            y = (double) (1 + j) / (double) (outData + 1);
            z = spline.Derivative(x, y, 2, 1);
            Console.Out.WriteLine(x.ToString("0.0000") + " y.ToString("0.0000") +" +F21(x, y).ToString("0.0000") +" +z.ToString("0.0000") +" +Math.Abs(F21(x, y) - z).ToString("0.0000")");
        }
    }
}
Example 4

This example integrates a two-dimensional, tensor-product spline over the rectangle \([0, x] \times [0, y]\).

```csharp
using System;
using Imsl.Math;

public class Spline2DInterpolateEx4
{
    // Define function
    private static double F(double x, double y)
    {
        return (x * x * x + y * y);
    }

    // The integral of F from 0 to x and 0 to y
    private static double FI(double x, double y)
    {
        return (y * x * x * x / 4.0 + x * y * y * y / 3.0);
    }

    public static void Main(String[] args)
    {
        int nData = 11, outData = 2;
        double[,] fData = new double[nData, nData];
        double[] xData = new double[nData];
        double[] yData = new double[nData];
        double x, y, z;

        // Set up grid
        for (int i = 0; i < nData; i++)
        {
            xData[i] = yData[i] = (double) i / ((double)(nData - 1));
        }

        for (int i = 0; i < nData; i++)
        {
            for (int j = 0; j < nData; j++)
            {
                fData[i, j] = F(xData[i], yData[j]);
            }
        }

        // Compute tensor-product interpolant
        Spline2DInterpolate spline =
            new Spline2DInterpolate(xData, yData, fData);
    }
}
```
```csharp
// Print results
Console.Out.WriteLine(" x  y  FI(x, y) Integral  Error");
for (int i = 0; i < outData; i++)
{
    x = (double) (1+i) / (double) (outData+1);
    for (int j = 0; j < outData; j++)
    {
        y = (double) (1+j) / (double) (outData+1);
        z = spline.Integral(0.0, x, 0.0, y);
        Console.Out.WriteLine(x.ToString("0.0000") + " "+
                              y.ToString("0.0000") + " "+
                              FI(x, y).ToString("0.0000") + " "+
                              z.ToString("0.0000") + " "+
                              Math.Abs(FI(x,y)-z).ToString("0.0000"));
    }
}
```

**Output**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>FI(x, y)</th>
<th>Integral</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3333</td>
<td>0.3333</td>
<td>0.0051</td>
<td>0.0051</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.3333</td>
<td>0.6667</td>
<td>0.0350</td>
<td>0.0350</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.6667</td>
<td>0.3333</td>
<td>0.0247</td>
<td>0.0247</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.6667</td>
<td>0.6667</td>
<td>0.0988</td>
<td>0.0988</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

**Spline2DLeastSquares Class**

```csharp
public class Imsl.Math.Spline2DLeastSquares : Spline2D

Computes a two-dimensional, tensor-product spline approximant using least squares.

The Spline2DLeastSquares class computes a tensor-product spline least-squares approximation to weighted tensor-product data. The input consists of data vectors to specify the tensor-product grid for the data, two vectors with the weights, the values of the surface on the grid, and the specification for the tensor-product spline. The grid is specified by the two vectors \( x = xData \) and \( y = yData \) of length \( n = xData.Length \) and \( m = yData.Length \), respectively. A two-dimensional array \( f = fData \) contains the data values which are to be fit. The two vectors \( w_x = xWeights \) and \( w_y = yWeights \) contain the weights for the weighted least-squares problem. The information for the approximating tensor-product spline can be provided using the SetXOrder, SetYOrder, SetXKnots and SetYKnots methods. This information is contained in \( k_x = xOrder \), \( t_x = xKnots \), and \( N = xSplineSpaceDim \) for the spline in the first variable, and in \( k_y = yOrder \), \( t_y = yKnots \) and \( M = ySplineSpaceDim \) for the spline in the second variable. This class computes coefficients for the tensor-product spline by solving the normal equations in tensor-product form as discussed in de Boor (1978, Chapter 17). The interested reader might also want to study the paper by Grosse (1980).```
As the computation proceeds, we obtain coefficients $c$ minimizing

$$
\left[ \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} w_x(i) w_y(j) \left( \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} c_{kl} B_{kl}(x_i, y_j) - f_{ij} \right) \right]^2
$$

where the function $B_{kl}$ is the tensor-product of two B-splines of order $k_x$ and $k_y$. Specifically, we have

$$
B_{kl}(x, y) = B_{k_x, k_x}(x) B_{k_y, k_y}(y)
$$

The spline

$$
\sum_{k=0}^{N-1} \sum_{l=0}^{M-1} c_{kl} B_{kl}
$$

and its partial derivatives can be evaluated using the Value method.

**Constructor**

**Spline2DLeastSquares**

```csharp
public Spline2DLeastSquares(double[] xData, double[] yData, double[,] fData, int xSplineSpaceDim, int ySplineSpaceDim)
```

**Description**

Constructor for Spline2DLeastSquares.

**Parameters**

- `xData` – A `double` array containing the data points in the $x$-direction.
- `yData` – A `double` array containing the data points in the $y$-direction.
- `fData` – A `double` matrix of size `xData.Length` by `yData.Length` containing the values to be approximated.
- `xSplineSpaceDim` – An `int` scalar value specifying the linear dimension of the spline subspace for the $x$ variable. It should be smaller than `xData.Length` and greater than or equal to `xOrder` (whose default value is 4).
- `ySplineSpaceDim` – An `int` scalar value specifying the linear dimension of the spline subspace for the $y$ variable. It should be smaller than `yData.Length` and greater than or equal to `yOrder` (whose default value is 4).

**Methods**

**Compute**

```csharp
public void Compute()
```
**Description**
Computes a two-dimensional, tensor-product spline approximant using least squares.

**GetErrorSumOfSquares**
public double GetErrorSumOfSquares()

**Description**
Returns the weighted error sum of squares.

**Returns**
A double scalar containing the weighted error sum of squares.

**GetXOrder**
virtual public int GetXOrder()

**Description**
Returns the order of the spline in the x-direction.

**Returns**
An int scalar containing the order of the spline in the x-direction.

**GetXWeights**
virtual public double[] GetXWeights()

**Description**
Returns the weights for the least-squares fit in the x-direction.

**Returns**
A double array containing the weights for the least-squares fit in the x-direction.

**GetYOrder**
virtual public int GetYOrder()

**Description**
Returns the order of the spline in the y-direction.

**Returns**
An int scalar containing the order of the spline in the y-direction.

**GetYWeights**
virtual public double[] GetYWeights()

**Description**
Returns the weights for the least-squares fit in the y-direction.

**Returns**
A double array containing the weights for the least-squares fit in the y-direction.

**SetXKnots**
virtual public void SetXKnots(double[] xKnots)
**Description**
Sets the knot sequences of the spline in the x-direction.

**Parameter**

- **xKnots** – A `double` array of size `xSplineSpaceDim + xOrder` specifying the knot sequences of the spline in the x-direction.
  
  Default: Knot sequences are selected by the class.

---

**SetXOrder**

```csharp
virtual public void SetXOrder(int xOrder)
```

**Description**
Sets the order of the spline in the x-direction.

**Parameter**

- **xOrder** – An `int` scalar value specifying the order of the spline in the x-direction. `xOrder` must be greater than or equal to 1.
  
  Default: `xOrder` = 4, implying a tensor-product cubic spline.

---

**SetXWeights**

```csharp
virtual public void SetXWeights(double[] xWeights)
```

**Description**
Sets the weights for the least-squares fit in the x-direction.

**Parameter**

- **xWeights** – A `double` array of size `xData.Length` specifying the weights for the least-squares fit in the x-direction.
  
  Default: All weights are equal to 1.

---

**SetYKnots**

```csharp
virtual public void SetYKnots(double[] yKnots)
```

**Description**
Sets the knot sequences of the spline in the y-direction.

**Parameter**

- **yKnots** – A `double` array of size `ySplineSpaceDim + yOrder` specifying the knot sequences of the spline in the y-direction.
  
  Default: Knot sequences are selected by the class.

---

**SetYOrder**

```csharp
virtual public void SetYOrder(int yOrder)
```

**Description**
Sets the order of the spline in the y-direction.
Parameter

\( y\text{Order} \) – An int scalar value specifying the order of the spline in the \( y \)-direction. \( y\text{Order} \) must be greater than or equal to 1.

Default: \( y\text{Order} = 4 \), implying a tensor-product cubic spline.

SetYWeights

**virtual** public void SetYWeights(double[] yWeights)

Description

Sets the weights for the least-squares fit in the \( y \)-direction.

Parameter

\( y\text{Weights} \) – A double array of size \( y\text{Data}.Length \) specifying the weights for the least-squares fit in the \( y \)-direction.

Default: All weights are equal to 1.

Example

The data for this example comes from the function \( e^x \sin(x + y) \) on the rectangle \([0, 3] \times [0, 5] \). This function is first sampled on a 50 x 25 grid. Next, an attempt to recover it by using tensor-product cubic splines is performed. The values of the function \( e^x \sin(x + y) \) are printed on a 2 x 2 grid and compared with the values of the tensor-product spline least-squares fit.

```csharp
using System;
using Imsl.Math;

public class Spline2DLeastSquaresEx1 {
    private static double F(double x, double y) {
        return (Math.Exp(x) * Math.Sin(x + y));
    }

    public static void Main(String[] args) {
        int nxData = 50, nyData = 25, outData = 2;
        double[] xData = new double[nxData];
        double[] yData = new double[nyData];
        double[,] fData = new double[nxData, nyData];
        double x, y, z;

        // Set up grid
        for (int i = 0; i < nxData; i++) {
            xData[i] = 3.0*(double) i / ((double) (nxData - 1));
        }
        for (int i = 0; i < nyData; i++) {
            yData[i] = 5.0*(double) i / ((double) (nyData - 1));
        }

        // Compute function values on grid
        for (int i = 0; i < nxData; i++) {
            for (int j = 0; j < nyData; j++) {
                fData[i, j] = F(xData[i], yData[j]);
            }
        }
    }
}
```
fData[i, j] = F(xData[i], yData[j]);
}

// Compute tensor-product approximant
Spline2DLeastSquares spline =
    new Spline2DLeastSquares(xData, yData, fData, 5, 7);
spline.Compute();
x = spline.GetErrorSumOfSquares();

// Print results
Console.Out.WriteLine("The error sum of squares is " +
    x.ToString("0.0000") + "\n");

double[,] output = new double[outData*outData, 5];
for (int i = 0, idx = 0; i < outData; i++) {
    x = (double) i / (double) (outData);
    for (int j = 0; j < outData; j++) {
        y = (double) j / (double) (outData);
        z = spline.Value(x, y);
        output[idx, 0] = x;
        output[idx, 1] = y;
        output[idx, 2] = F(x, y);
        output[idx, 3] = z;
        output[idx, 4] = Math.Abs(F(x, y) - z);
        idx++;
    }
}

String[] labels = {"x", "y", "F(x, y)", "Fitted Values", "Error"};
PrintMatrixFormat pmf = new PrintMatrixFormat();
    pmf.NumberFormat = "0.0000";
    pmf.SetColumnLabels(labels);
new PrintMatrix().Print(pmf, output);

Output
The error sum of squares is 3.7532

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F(x, y)</th>
<th>Fitted Values</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>-0.0204</td>
<td>0.0204</td>
</tr>
<tr>
<td>1</td>
<td>0.0000</td>
<td>0.5000</td>
<td>0.4794</td>
<td>0.5002</td>
</tr>
<tr>
<td>2</td>
<td>0.5000</td>
<td>0.0000</td>
<td>0.7904</td>
<td>0.8158</td>
</tr>
<tr>
<td>3</td>
<td>0.5000</td>
<td>0.5000</td>
<td>1.3874</td>
<td>1.3842</td>
</tr>
</tbody>
</table>
RadialBasis Class

public class Imsl.Math.RadialBasis

RadialBasis computes a least-squares fit to scattered data in \( \mathbb{R}^d \), where \( d \) is the dimension. More precisely, we are given data points

\[
x_0, \ldots, x_{n-1} \in \mathbb{R}^d
\]

and function values

\[
f_0, \ldots, f_{n-1} \in \mathbb{R}^1
\]

The radial basis fit to the data is a function \( F \) which approximates the above data in the sense that it minimizes the sum-of-squares error

\[
\sum_{i=0}^{n-1} w_i (F(x_i) - f_i)^2
\]

where \( w \) are the weights. Of course, we must restrict the functional form of \( F \). Here we assume it is a linear combination of radial functions:

\[
F(x) \equiv \sum_{j=0}^{m-1} \alpha_j \phi(\|x - c_j\|)
\]

The \( c_j \) are the centers.

A radial function, \( \phi(r) \), maps \([0, \infty)\) into \( \mathbb{R}^1 \). The default radial function is the Hardy multiquadric,

\[
\phi(r) \equiv \sqrt{r^2 + \delta^2}
\]

with \( \delta = 1 \). An alternate radial function is the Gaussian, \( e^{-ar^2} \).

By default, the centers are points in a Faure sequence, scaled to cover the box containing the data.

Two Update methods allow the user to specify weights for each data point in the approximation scheme. In this way the user can influence the fit of the radial basis function. For example, if weights are in the range \([0,1]\) then 0-weighted points are effectively removed from computations and 1-weighted points will have more influence than any others. When the number of centers equals the number of data points, the RBF fit will be “exact”, otherwise it will be an approximation (useful for large or noisy data sets). Provided the ratios of the weights are not too extreme, weights will not appreciably change the accuracy of the fit to the data, but they will affect the shape of the approximating function away from the data: Greater weights result in greater influence at greater distances.

Properties

ANOVA

public Imsl.Stat.ANOVA ANOVA {get; }

Interpolation and Approximation
Description
Returns the ANOVA statistics from the linear regression.

Property Value
An ANOVA table and related statistics.
See Also: Imsl.Stat.LinearRegression (p. 634), Imsl.Stat.ANOVA (p. 708)

RadialFunction
public Imsl.Math.RadialBasis.IFunction RadialFunction {get; set; }

Description
The radial function.

Property Value
A RadialBasis.IFunction which is the current radial function.

Constructor

RadialBasis
public RadialBasis(int nDim, int nCenters)

Description
Creates a new instance of RadialBasis.

Parameters
  nDim – An int specifying the number of dimensions.
  nCenters – An int specifying the number of centers.

Methods

Eval
public double Eval(double[] x)

Description
Returns the value of the radial basis approximation at a point.

Parameter
  x – A double array containing the locations of the data point at which the approximation is to be computed.
Returns
A double containing the value of the radial basis approximation at x.

Eval
public double[] Eval(double[,] x)

Description
Returns the value of the radial basis at a point.

Parameter
x – A double matrix of size n by nDim containing the points at which the radial basis is to be evaluated.

Returns
A double array giving the value of the radial basis at the point x

Gradient
public double[] Gradient(double[] x)

Description
Returns the gradient of the radial basis approximation at a point.

Parameter
x – A double array containing the locations of the data point at which the approximation’s gradient is to be computed.

Returns
A double array, of length nDim containing the value of the gradient of the radial basis approximation at x.

Update
public void Update(double[] x, double f)

Description
Adds a data point with weight = 1.

Parameters
x – A double array containing the locations of the data point.
  f – A double containing the function value at the data point.

Update
public void Update(double[] x, double f, double w)

Description
Adds a data point with a specified weight.
Parameters
x – A double array containing the locations of the data point.
f – A double containing the function value at the data point.
w – A double containing the weight of this data point.

Update
public void Update(double[,] x, double[] f)

Description
Adds a set of data points, all with weight = 1.

Parameters
x – A double matrix of size $n$ by $nDim$ containing the locations of the data points for each dimension.
f – A double array containing the function values at the data points.

Update
public void Update(double[,] x, double[] f, double[] w)

Description
Adds a set of data points with user-specified weights.

Parameters
x – A double matrix of size $n$ by $nDim$ containing the locations of the data points for each dimension.
f – A double array containing the function values at the data points.
w – A double array containing the weights associated with the data points.

Example: Radial Basis Function Approximation
The function

$$e^{-\|\mathbf{x}\|^2/d}$$

where $d$ is the dimension, is evaluated at a set of randomly chosen points. Random noise is added to the values and a radial basis approximated to the noisy data is computed. The radial basis fit is then compared to the original function at another set of randomly chosen points. Both the average error and the maximum error are computed and printed.

In this example, the dimension $d=10$. The function is sampled at 200 random points, in the $[-1,1]^d$ cube, to which what noise in the range [-0.2,0.2] is added. The error is computed at 1000 random points, also from the $[-1,1]^d$ cube. The compute errors are less than the added noise.

```csharp
using System;
using Imsl.Math;

public class RadialBasisEx1
{
```
public static void Main(String[] args)
{
    int nDim = 10;

    // Sample, with noise, the function at 100 randomly chosen points
    int nData = 200;
    double[,] xData = new double[nData, nDim];
    double[] fData = new double[nData];
    double[] tmp = new double[nDim];
    for (int k = 0; k < nData; k++)
    {
        for (int i = 0; i < nDim; i++)
        {
            tmp[i] = xData[k, i] = 2.0 * rand.NextDouble() - 1.0;
        }
        // noisy sample
        fData[k] = fcn(tmp) + 0.20 * (2.0 * rand.NextDouble() - 1.0);
    }

    // Compute the radial basis approximation using 25 centers
    int nCenters = 25;
    RadialBasis rb = new RadialBasis(nDim, nCenters);
    rb.Update(xData, fData);

    // Compute the error at a randomly selected set of points
    int nTest = 1000;
    double maxError = 0.0;
    double aveError = 0.0;
    double[] x = new double[nDim];
    for (int k = 0; k < nTest; k++)
    {
        for (int i = 0; i < nDim; i++)
        {
            x[i] = 2.0 * rand.NextDouble() - 1.0;
        }
        double error = Math.Abs(fcn(x) - rb.Eval(x));
        aveError += error;
        maxError = Math.Max(error, maxError);
        double f = fcn(x);
    }
    aveError /= nTest;
    Console.WriteLine("average error is " + aveError);
    Console.WriteLine("maximum error is " + maxError);
}

// The function to approximate
internal static double fcn(double[] x)
{
    double sum = 0.0;
    for (int k = 0; k < x.Length; k++)
    {
        sum += x[k] * x[k];
    }
average error is 0.0419789795502543
maximum error is 0.171666811944547

Example: “Custom” Radial Basis Function Approximation

Data is generated from the function

$$e^{\frac{xy}{2}} \sin x \cos \frac{y}{2.0}$$

where a number of \((x,y)\) pairs make up a set of randomly chosen points. Random noise is added to the values, a “custom” Polyharmonic Spline radial basis function is specified

$$\varphi(r) = \begin{cases} r^k & k = 1, 3, 5, \ldots \\ r^k \ln r & k = 2, 4, 6, \ldots \end{cases}$$

and a radial basis approximation of the noisy data is computed. The radial basis fit is then compared to the original function at another set of randomly chosen points. Both the average normalized error and the maximum normalized error are computed and printed.

In this example, the order of the Polyharmonic Spline, \(k=2\). The function is sampled at 200 random points and the error is computed at 10000 random points.
virtual public bool EvenOrder
{
    get
    {
        return isEven;
    }
}

private int order = 3;
private bool isEven = false;

public PolyHarmonicSpline(int order)
{
    this.isEven = order % 2 == 0;
    this.order = order;
}

public virtual double F(double x)
{
    if (this.isEven)
    {
        return Math.Pow(x, order) * Math.Log(x);
    }
    return Math.Pow(x, order);
}

public virtual double G(double x)
{
    if (order == 1)
    {
        return 1;
    }
    if (this.isEven)
    {
        return order * Math.Pow(x, order - 1) * Math.Log(x) +
        Math.Pow(x, order - 1);
    }
    return order * Math.Pow(x, order - 1);
}

public static void Main(String[] args)
{
    int nDim = 2;
    // Sample, with noise, the function at 100 randomly chosen points
    int nData = 200;
    double[,] xData = new double[nData, nDim];
    double[] fData = new double[nData];
    double[] row = new double[nDim];
    rand.Multiplier = 16807;
    double[] noise = new double[nData * nDim];
    for (int k = 0; k < nData; k++)
    {
for (int i = 0; i < nDim; i++)
{
    noise[k * 2 + i] = 1.0d - 2.0d * rand.NextDouble();
    xData[k, i] = 3 * noise[k * 2 + i];
}

for(int j = 0; j<nDim; j++)
    row[j]=xData[k,j];

    fData[k] = fcn(row) + noise[k * 2] / 10;

for (int i = 0; i < nDim; i++)
{
    noise[i * 2 + j] = 1.0d - 2.0d * (double) rand.NextDouble();
    x[i][j] = 3 * noise[i * 2 + j];
}

double error = Math.Abs(fcn(x[i]) - rb.Eval(x[i]));
maxMagnitude = Math.Max(Math.Abs(fcn(x[i])), maxMagnitude);
aveError += error;
maxError = Math.Max(error, maxError);
}

aveError /= nTest;

Console.WriteLine("Average normalized error is " + aveError / maxMagnitude);
Console.WriteLine("Maximum normalized error is " + maxError / maxMagnitude);
Console.WriteLine("Using even order equation: " + ((PolyHarmonicSpline) rb.RadialFunction).EvenOrder);
Example: Multiquadric Radial Basis Function Approximation

Data is generated from the function
\[ e^{\frac{y}{2}} \sin x \cos \frac{y}{2.0} \]
where a number of \((x,y)\) pairs make up a set of randomly chosen points. Random noise is added to the values, a Hardy multiquadric radial basis function is specified \(\sqrt{r^2 + \delta^2}\) and a radial basis approximation of the noisy data is computed. The radial basis fit is then compared to the original function at another set of randomly chosen points. Both the average normalized error and the maximum normalized error are computed and printed.

In this example, the parameter of the Hardy multiquadric radial basis function \(\delta = 5.5\). The function is sampled at 100 random points and the error is computed at 10000 random points.

```csharp
using System;
using Imsl.Math;

public class RadialBasisEx3
{
    public static void Main(String[] args)
    {
        int nDim = 2;

        // Sample, with noise, the function at 100 randomly chosen points
        int nData = 100;
        double[,] xData = new double[nData, nDim];
        double[] fData = new double[nData];
        double[] row = new double[nDim];
        rand.Multiplication = 16807;
        double[] noise = new double[nData * nDim];
        for (int k = 0; k < nData; k++)
        {
            for (int i = 0; i < nDim; i++)
            {
                noise[k * 2 + i] = 1.0d - 2.0d * (double) rand.NextDouble();
                xData[k, i] = 3 * noise[k * 2 + i];
            }
        }
        // noisy sample
        for(int j = 0; j<nDim; j++)
        { row[j]=xData[k,j];
            fData[k] = fcn(row) + noise[k*2] / 10;
        }
        // Compute the radial basis approximation using 100 centers
        int nCenters = 100;
```
RadialBasis rb = new RadialBasis(nDim, nCenters);
rb.RadialFunction = new RadialBasis.HardyMultiquadric(5.5);
rb.Update(xData, fData);

// Compute the error at a randomly selected set of points
int nTest = 10000;
double maxError = 0.0;
double aveError = 0.0;
double maxMagnitude = 0.0;
Imsl.WarningObject w = Imsl.Warning.WarningObject;
Imsl.Warning.WarningObject = null;
double[][] x = new double[nTest][];
for (int i2 = 0; i2 < nTest; i2++)
{
    x[i2] = new double[nDim];
}
noise = new double[nTest * nDim];

for (int i = 0; i < nTest; i++)
{
    for (int j = 0; j < nDim; j++)
    {
        noise[i * 2 + j] = 1.0d - 2.0d * rand.NextDouble();
        x[i][j] = 3 * noise[i * 2 + j];
    }
    double error = Math.Abs(fcn(x[i]) - rb.Eval(x[i]));
    maxMagnitude = Math.Max(Math.Abs(fcn(x[i])),
                            maxMagnitude);
    aveError += error;
    maxError = Math.Max(error, maxError);
}
aveError /= nTest;
Imsl.Warning.WarningObject = w;
Console.WriteLine("Average normalized error is " +
                    aveError / maxMagnitude);
Console.WriteLine("Maximum normalized error is " +
                    maxError / maxMagnitude);

// The function to approximate
internal static double fcn(double[] x)
{
    return Math.Exp((x[1]) / 2.0) * Math.Sin(x[0]) -
            Math.Cos((x[1]) / 2.0);
}

Output

Average normalized error is 0.0110861431448538
Maximum normalized error is 0.054260728660165
Example: Gaussian Radial Basis Function Approximation

Data is generated from the function
\[ e^{\frac{\sqrt{x^2 + y^2}}{2.0}} \sin x \cos \frac{y}{2.0} \]

where a number of \((x, y)\) pairs make up a set of randomly chosen points. Random noise is added to the values, a Gaussian radial basis function is specified \(e^{-ax^2}\) and a radial basis approximation of the noisy data is computed. The radial basis fit is then compared to the original function at another set of randomly chosen points. Both the average normalized error and the maximum normalized error are computed and printed.

In this example, the parameter of the Gaussian radial basis function \(a = 0.1\). The function is sampled at 100 random points and the error is computed at 10000 random points.

```csharp
using System;
using Imsl.Math;

class RadialBasisEx4
{
    static void Main(string[] args)
    {
        int nDim = 2;

        // Sample, with noise, the function at 100 randomly chosen points
        int nData = 100;
        double[,] xData = new double[nData, nDim];
        double[] fData = new double[nData];
        double[] row = new double[nDim];
        rand.Multipier = 16807;
        double[] noise = new double[nData * nDim];
        for (int k = 0; k < nData; k++)
        {
            for (int i = 0; i < nDim; i++)
            {
                noise[k * 2 + i] = 1.0d - 2.0d * (double) rand.NextDouble();
                xData[k, i] = 3 * noise[k * 2 + i];
            }
            // noisy sample
            for (int j = 0; j < nDim; j++)
            {
                row[j] = xData[k, j];
                fData[k] = fcn(row) + noise[k * 2] / 10;
            }
        }

        // Compute the radial basis approximation using 100 centers
        int nCenters = 100;
        RadialBasis rb = new RadialBasis(nDim, nCenters);
        rb.RadialFunction = new RadialBasis.Gaussian(.1);
        rb.Update(xData, fData);

        // Compute the error at a randomly selected set of points
        int nTest = 10000;
    }
}
```
double maxError = 0.0;
double aveError = 0.0;
double maxMagnitude = 0.0;
Imsl.WarningObject w = Imsl.Warning.WarningObject;
Imsl.Warning.WarningObject = null;
double[,] x = new double[nTest,];
for (int i2 = 0; i2 < nTest; i2++)
{
    x[i2] = new double[nDim];
}
noise = new double[nTest * nDim];
for (int i = 0; i < nTest; i++)
{
    for (int j = 0; j < nDim; j++)
    {
        noise[i * 2 + j] = 1.0d - 2.0d * rand.NextDouble();
        x[i][j] = 3 * noise[i * 2 + j];
    }
    double error = Math.Abs(fcn(x[i]) - rb.Eval(x[i]));
    maxMagnitude = Math.Max(Math.Abs(fcn(x[i])),
        maxMagnitude);
    aveError += error;
    maxError = Math.Max(error, maxError);
}
aveError /= nTest;
Imsl.Warning.WarningObject = w;
Console.WriteLine("Average normalized error is "+
    aveError / maxMagnitude);
Console.WriteLine("Maximum normalized error is "+
    maxError / maxMagnitude);

// The function to approximate
internal static double fcn(double[] x)
{
    return Math.Exp((x[1]) / 2.0) * Math.Sin(x[0]) -
        Math.Cos((x[1]) / 2.0);
}

Output
Average normalized error is 0.0109549594616543
Maximum normalized error is 0.0230039785369829
RadialBasis.IFunction Interface

public interface Imsl.Math.RadialBasis.IFunction
Public interface for the user supplied function to the RadialBasis object.

Methods

F
abstract public double F(double x)

Description
A radial basis function.

Parameter
x – A double, the point at which the function is to be evaluated.

Returns
A double, the value of the function at x.

G
abstract public double G(double x)

Description
The derivative of the radial basis function used to calculate the gradient of the radial basis approximation.

Parameter
x – A double, the point at which the function is to be evaluated.

Returns
A double, the value of the function at x.

RadialBasis.Gaussian Class

The Gaussian basis function, $e^{-ax^2}$. 

Interpolation and Approximation  RadialBasis.IFunction • 207
Constructor

**Gaussian**

public Gaussian(double a)

Description

Creates a Gaussian basis function $e^{-ax^2}$.

Parameter

a – A double specifying the value of the function parameter. Decreasing the Gaussian parameter decreases fitting-error but may increase computational effort.

Methods

**F**

Final public double F(double x)

Description

A Gaussian basis function.

Parameter

x – A double, the point at which the function is to be evaluated.

Returns

A double, the value of the function at x.

**G**

Final public double G(double x)

Description

The derivative of the Gaussian basis function used to calculate the gradient of the radial basis approximation.

Parameter

x – A double, the point at which the function is to be evaluated.

Returns

A double, the value of the function at x.
RadialBasis.HardyMultiquadric Class


The Hardy multiquadric basis function, $\sqrt{r^2 + \delta^2}$.

Constructor

HardyMultiquadric

public HardyMultiquadric(double delta)

Description

Creates a Hardy multiquadric basis function $\sqrt{r^2 + \delta^2}$.

Parameter

delta – A double specifying the value of the function parameter. Increasing the multiquadric parameter decreases fitting-error but generally increases computational effort.

Default: delta = 1.0.

Methods

F

Final public double F(double x)

Description

A Hardy multiquadric basis function.

Parameter

x – A double, the point at which the function is to be evaluated.

Returns

A double, the value of the function at x.

G

Final public double G(double x)

Description

The derivative of the Hardy multiquadric basis function used to calculate the Gradient of the radial basis approximation.
Parameter
x – A double, the point at which the function is to be evaluated.

Returns
A double, the value of the function at x.
Chapter 4: Quadrature

Types

class Quadrature .............................................................. 212
interface Quadrature.IFunction ........................................... 219
class HyperRectangleQuadrature .......................................... 220
interface HyperRectangleQuadrature.IFunction ....................... 224

Usage Notes

Univariate Quadrature

Class Quadrature computes approximations to integrals of the form

\[ \int_{c}^{b} f(x) \, dx \]

One situation that occasionally arises in univariate quadrature concerns the approximation of integrals when only tabular data are given. The functions described above do not directly address this question. However, the standard method for handling this problem is first to interpolate the data, and then to integrate the interpolant. This can be accomplished by using an IMSL C# Library spline interpolation class derived from Imsl.Math.Spline and the method Imsl.Math.Spline.Integral(a,b)

Multivariate Quadrature

The class HypercubeQuadrature computes an approximation to the integral of a function of \( n \) variables over a hyper-rectangle.

\[ \int_{d_1}^{b_1} \ldots \int_{d_n}^{b_n} f(x_1, \ldots, x_n) \, dx_n \ldots dx_1 \]

Return Values from User-Supplied Functions

All values returned by user-supplied functions must be valid real numbers. It is the user’s responsibility to check that the values returned by a user-supplied function do not contain NaN, infinity, or negative infinity values.
Example: Minimum of a smooth function

The minimum of $e^x - 5x$ is found using function evaluations only.

```csharp
using System;
using Imsl.Math;

public class OptimizationIntroEx1 : MinUncon.IFunction
{
    public double F(double x)
    {
        double y = Math.Exp(x) - 5.0 * x;
        if (!(Double.IsNaN(y)))
        {
            return y;
        } else
        {
            return 0.0;
        }
    }

    public static void Main(String[] args)
    {
        MinUncon zf = new MinUncon();
        zf.Guess = 0.0;
        zf.Accuracy = 0.001;
        MinUncon.IFunction fcn = new OptimizationIntroEx1();
        Console.Write("Minimum is " + zf.ComputeMin(fcn));
    }
}
```

Quadrature Class

```csharp
public class Imsl.Math.Quadrature

Quadrature is a general-purpose integrator that uses a globally adaptive scheme in order to reduce the absolute error.

Quadrature subdivides the interval $[A, B]$ and uses a $(2k + 1)$-point Gauss-Kronrod rule to estimate the integral over each subinterval. The error for each subinterval is estimated by comparison with the $k$-point Gauss quadrature rule. The subinterval with the largest estimated error is then bisected and the same procedure is applied to both halves. The bisection process is continued until either the error criterion is satisfied, roundoff error is detected, the subintervals become too small, or the maximum number of subintervals allowed is reached. The Class Quadrature is based on the subroutine QAG by Piessens et al. (1983).

If the function to be integrated has endpoint singularities then extrapolation should be enabled. As described above, the integral’s value is approximated by applying a quadrature rule to a series of
```
subdivisions of the interval. The sequence of approximate values converges to the integral’s value. The
\( \varepsilon \)-algorithm can be used to extrapolate from the initial terms of the sequence to its limit. Without
extrapolation, the quadrature approximation sequence converges slowly if the function being integrated
has endpoint singularities. The \( \varepsilon \)-algorithm accelerates convergence of the sequence in this case. The
class EpsilonAlgorithm (p. 513) implements the \( \varepsilon \)-algorithm. With extrapolation, this class is similar to
the subroutine QAGS by Piessens et al. (1983).

The desired absolute error, \( \varepsilon \), can be set using AbsoluteError property. The desired relative error, \( \rho \),
can be set using RelativeError property. The method Eval computes the approximate integral value
\( R \approx \int_a^b f(x)dx \). It also computes an error estimate \( E \), which can be retrieved using ErrorEstimate
property. These are related by the following equation:

\[
\left| \int_a^b f(x)dx - R \right| \leq E \leq \max \left\{ \varepsilon, \rho \left| \int_a^b f(x)dx \right| \right\}
\]

### Properties

**AbsoluteError**

```csharp
public double AbsoluteError { get; set; }
```

**Description**
The absolute error tolerance.

**Property Value**
A double scalar value specifying the absolute error.

**ErrorEstimate**

```csharp
public double ErrorEstimate { get; }
```

**Description**
Returns an estimate of the relative error in the computed result.

**Property Value**
A double specifying an estimate of the relative error in the computed result.

**ErrorStatus**

```csharp
public int ErrorStatus { get; }
```

**Description**
Returns the non-fatal error status.

**Property Value**
An int specifying the non-fatal error status:
### Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error.</td>
</tr>
<tr>
<td>1</td>
<td>Maximum number of subdivisions allowed has been achieved. One can allow more subdivisions by setting the <code>MaxSubintervals</code>. If this yields no improvement it is advised to analyze the integrand in order to determine the integration difficulties. If the position of a local difficulty can be determined (e.g. singularity, discontinuity within the interval) one will probably gain from splitting up the interval at this point and calling the integrator on the subranges. If possible, an appropriate special-purpose integrator should be used, which is designed for handling the type of difficulty involved.</td>
</tr>
<tr>
<td>2</td>
<td>The occurrence of roundoff error is detected, which prevents the requested tolerance from being achieved. The error may be underestimated.</td>
</tr>
<tr>
<td>3</td>
<td>Extremely bad integrand behavior occurs at some points of the integration interval.</td>
</tr>
<tr>
<td>4</td>
<td>The algorithm does not converge. Roundoff error is detected in the extrapolation table. It is presumed that the requested tolerance cannot be obtained.</td>
</tr>
<tr>
<td>5</td>
<td>The algorithm does not converge. Roundoff error is detected in the extrapolation table. It is presumed that the requested tolerance cannot be achieved, and that the returned result is the best which can be obtained.</td>
</tr>
<tr>
<td>6</td>
<td>The integral is probably divergent, or slowly convergent. It must be noted that divergence can occur with any other status value.</td>
</tr>
</tbody>
</table>

### Extrapolation

```csharp
public bool Extrapolation {get; set; }
```

**Description**

If `true`, the epsilon-algorithm for extrapolation is enabled.

**Property Value**

A `bool`, `true` if the epsilon-algorithm for extrapolation is to be enabled, `false` otherwise.

**Remarks**

The default is `false` (extrapolation is not used).

### MaxSubintervals

```csharp
public int MaxSubintervals {get; set; }
```

**Description**

The maximum number of subintervals allowed.

**Property Value**

An `int` specifying the maximum number of subintervals to be allowed.
Remarks
The default value is 500.

**NumberOfProcessors**

```csharp
public int NumberOfProcessors {get; set; }
```

Description
Perform the parallel calculations with the maximum possible number of processors set to
NumberOfProcessors.

Property Value
An int indicating the maximum possible number of processors to use.

Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is
set to a number less than 1 or greater than Environment.ProcessorCount,
Environment.ProcessorCount will be used internally. This property has no effect if the application is
used with a scalar version of the IMSL C# Numerical Library.

**Parallel**

```csharp
public bool Parallel {get; set; }
```

Description
Enable or disable performing Quadrature.IFunction.F in parallel.

Property Value
A bool indicating whether or not the Quadrature.IFunction.F calculations are to be performed in
parallel.

Remarks
By default, Parallel = true. This property has no effect if the application is used with a scalar version
of the IMSL C# Numerical Library.

**RelativeError**

```csharp
public double RelativeError {get; set; }
```

Description
The relative error tolerance.

Property Value
A double scalar value specifying the relative error.

**Rule**

```csharp
public int Rule {get; set; }
```

Description
The Gauss-Kronrod rule.

Property Value
An int specifying the rule to be used.
**Remarks**
The default is 3.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Data points used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7 - 15</td>
</tr>
<tr>
<td>2</td>
<td>10 - 21</td>
</tr>
<tr>
<td>3</td>
<td>15 - 31</td>
</tr>
<tr>
<td>4</td>
<td>20 - 41</td>
</tr>
<tr>
<td>5</td>
<td>25 - 51</td>
</tr>
<tr>
<td>6</td>
<td>30 - 61</td>
</tr>
</tbody>
</table>

**Constructor**

**Quadrature**

public Quadrature()

**Description**

Constructs a Quadrature object.

**Method**

**Eval**

public double Eval(Imsl.Math.Quadrature.IFunction f, double a, double b)

**Description**

Returns the value of the integral from a to b.

**Parameters**

- **f** – The function to be integrated.
- **a** – A double specifying the lower limit of integration.
- **b** – A double specifying the upper limit of integration, either or both of a and b can be Double.POSITIVE_INFINITY or Double.NEGATIVE_INFINITY.

**Returns**

A double specifying the integral value from a to b.

**Example 1: Integral of exp(2x)**

The integral $\int_{1}^{3} e^{2x} dx$ is computed and compared to its expected value.
using System;
using Imsl.Math;

public class QuadratureEx1 : Quadrature.IFunction
{
    public double F(double x)
    {
        return Math.Exp(2.0 * x);
    }

    public static void Main(String[] args)
    {
        Quadrature q = new Quadrature();
        Quadrature.IFunction fcn = new QuadratureEx1();
        double result = q.Eval(fcn, 1.0, 3.0);

        double expect =
            (System.Math.Exp(6) - System.Math.Exp(2)) / 2.0;
        Console.Out.WriteLine("result = " + result);
        Console.Out.WriteLine("expect = " + expect);
    }
}

Output
result = 198.019868696902
expect = 198.019868696902

Example 2: Integral of exp(-x) from 0 to infinity
The integral \( \int_0^\infty e^{-x} \, dx \) is computed and compared to its expected value.

using System;
using Imsl.Math;

public class QuadratureEx2 : Quadrature.IFunction
{
    public double F(double x)
    {
        return Math.Exp(-x);
    }

    public static void Main(String[] args)
    {
        Quadrature q = new Quadrature();
        Quadrature.IFunction fcn = new QuadratureEx2();
        double result = q.Eval(fcn, 0.0, Double.PositiveInfinity);

        double expect = 1.0;
        Console.Out.WriteLine("result = " + result);
        Console.Out.WriteLine("expect = " + expect);
    }
}
Example 3: Integral of the entire real line

The integral \( \int_{-\infty}^{\infty} \frac{x^4 e^x + 9 e^{-x}}{4e^x + 9e^{-x}} \, dx \) is computed and compared to its expected value. This integral is evaluated in Gradshteyn and Ryzhik (equation 3.417.1).

```csharp
using System;
using Imsl.Math;

public class QuadratureEx3 : Quadrature.IFunction
{
    public double F(double x)
    {
        return x / (4.0 * Math.Exp(x) + 9.0 * Math.Exp(-x));
    }

    public static void Main(String[] args)
    {
        Quadrature q = new Quadrature();
        Quadrature.IFunction fcn = new QuadratureEx3();
        double result = q.Eval(fcn, Double.NegativeInfinity, Double.PositiveInfinity);
        double expect = System.Math.PI * System.Math.Log(1.5) / 12.0;
        Console.Out.WriteLine("result = " + result);
        Console.Out.WriteLine("expect = " + expect);
    }
}
```

Output

result = 0.106150517076628
expect = 0.106150517076633

Reference


Example 4: Integral of an oscillatory function

The integral of \( \cos(ax) \) for \( a = 10^4 \) is computed and compared to its expected value. Because the function is highly oscillatory, the quadrature rule is set to 6. The relative error tolerance is also set.

```csharp
using System;

218 • Quadrature  IMSL C# Numerical Library
```
using Imsl.Math;

public class QuadratureEx4 : Quadrature.IFunction
{
    private double a;

    public QuadratureEx4(double a)
    {
        this.a = a;
    }

    public double F(double x)
    {
        return Math.Cos(a * x);
    }

    public static void Main(String[] args)
    {
        double a = 1.0e4;
        Quadrature.IFunction fcn = new QuadratureEx4(a);

        Quadrature q = new Quadrature();
        q.Rule = 6;
        q.RelativeError = 1e-10;
        double result = q.Eval(fcn, 0.0, 1.0);

        double expect = Math.Sin(a) / a;
        Console.Out.WriteLine("result = " + result);
        Console.Out.WriteLine("expect = " + expect);
        Console.Out.WriteLine("relative error = " + (expect - result) / expect);
        Console.Out.WriteLine("relative error estimate = " + q.ErrorEstimate);
    }
}

Output
result = -3.05614388902539E-05
expect = -3.05614388888252E-05
relative error = -4.67475872674725E-11
relative error estimate = 1.04883755430041E-08

Quadrature.IFunction Interface

public interface Imsl.Math.Quadrature.IFunction

Interface defining function for the Quadrature class.
Method

**F**

abstract public double F(double x)

**Description**
Function to be integrated.

**Parameter**

- x – A double specifying the point at which the function is to be evaluated.

**Returns**

A double specifying the value of the function at x.

---

**HyperRectangleQuadrature Class**

```csharp
public class Imsl.Math.HyperRectangleQuadrature

HyperRectangleQuadrature integrates a function over a hypercube.

This class is used to evaluate integrals of the form:

\[
\int_{a_0}^{b_0} \cdots \int_{a_{n-1}}^{b_{n-1}} f(x_0, \ldots, x_{n-1}) \, dx_0 \cdots dx_{n-1}
\]

Integration of functions over hypercubes by Monte Carlo, in which the integral is evaluated as the value of the function averaged over a sequence of randomly chosen points. Under mild assumptions on the function, this method will converge like 1/\sqrt{n}, where n is the number of points at which the function is evaluated.

It is possible to improve on the performance of Monte Carlo by carefully choosing the points at which the function is to be evaluated. Randomly distributed points tend to be non-uniformly distributed. The alternative to a sequence of random points is a low-discrepancy sequence. A low-discrepancy sequence is one that is highly uniform.

This function is based on the low-discrepancy Faure sequence as computed by Imsl.Stat.FaureSequence (p. 1216).

**Properties**

**AbsoluteError**

```csharp
public double AbsoluteError {get; set; }
```
Sets the absolute error tolerance.

**Property Value**
A double scalar value specifying the absolute error tolerance.

**ErrorEstimate**
public double ErrorEstimate {get; }

Returns an estimate of the relative error in the computed result.

**Property Value**
A double specifying an estimate of the relative error in the computed result.

**NumberOfProcessors**
public int NumberOfProcessors {get; set; }

Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**
An int indicating the maximum possible number of processors to use.

**Remarks**
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**Parallel**
public bool Parallel {get; set; }

Enable or disable performing HyperRectangleQuadrature(IFunction.F in parallel.

**Property Value**
A bool indicating whether or not the HyperRectangleQuadrature(IFunction.F calculations are to be performed in parallel.

**Remarks**
By default, Parallel = true. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**RelativeError**
public double RelativeError {get; set; }
Description
Sets the relative error tolerance.

Property Value
A double scalar value specifying the relative error tolerance.

Constructors

HyperRectangleQuadrature
public HyperRectangleQuadrature(int dimension)

Description
Constructs a HyperRectangleQuadrature object.

Parameter
dimension – A int which specifies the dimension of the Faure sequence.

HyperRectangleQuadrature
public HyperRectangleQuadrature(Imsl.Stat.IRandomSequence sequence)

Description
Constructs a HyperRectangleQuadrature object.

Parameter
sequence – A IRandomSequence object containing the random number sequence.

Methods

Eval
public double Eval(Imsl.Math.HyperRectangleQuadrature.IFunction f)

Description
Returns the value of the integral over the unit cube.

Parameter
f – A IFunction containing the function to be integrated.

Returns
A double containing the value of the integral over the unit cube.
Description
Returns the value of the integral over a cube.

Parameters
f – A IFunction containing the function to be integrated.
   a – A double specifying the lower limit of integration. If null all of the lower limits default to 0.
   b – A double specifying the upper limit of integration. If null all of the upper limits default to 1.

Returns
A double containing the value of the integral over the unit cube.

Example: HyperRectangle Quadrature
This example evaluates the following multidimensional integral, with $n=10$.

$$\int_{b_n-1}^{a_n-1} \cdots \int_{a_0}^{b_0} \left[ \sum_{i=0}^{n} (-1)^i \prod_{j=0}^{i} x_j \right] dx_0 \cdots dx_{n-1} = \frac{1}{3} \left[ 1 - \left(-\frac{1}{2}\right)^n \right]$$

using System;
using Imsl.Math;

public class HyperRectangleQuadratureEx1 : HyperRectangleQuadrature.IFunction
{
    public double F(double[] x)
    {
        int sign = 1;
        double sum = 0.0;
        for (int i = 0; i < x.Length; i++)
        {
            double prod = 1.0;
            for (int j = 0; j <= i; j++)
            {
                prod *= x[j];
            }
            sum += sign * prod;
            sign = - sign;
        }
        return sum;
    }

    public static void Main(String[] args)
    {
        HyperRectangleQuadrature q = new HyperRectangleQuadrature(10);
        double result = q.Eval(new HyperRectangleQuadratureEx1());
        Console.Out.WriteLine("result = " + result);
    }
}

Output
result = 0.333125383208954
HyperRectangleQuadrature.IFunction Interface

public interface Imsl.Math.HyperRectangleQuadrature.IFunction
Interface for the HyperRectangleQuadrature function.

Method

F
abstract public double F(double[] x)

Description
Returns the value of the function at the given point.

Parameter
x – A double array specifying the point at which the function is to be evaluated.

Returns
A double specifying the value of the function at x.
Chapter 5: Differential Equations

Types

class ODE

enumeration ODE.ExamineStepOptions

enumeration ODE.ErrorNormOptions

class OdeRungeKutta

interface OdeRungeKutta.IFunction

class OdeAdamsGear

enumeration OdeAdamsGear.IntegrationType

enumeration OdeAdamsGear.SolveOption

interface OdeAdamsGear.IFunction

interface OdeAdamsGear.IJacobian

class FeynmanKac

enumeration FeynmanKac.PDEStepControlMethod

interface FeynmanKac.IPdeCoefficients

interface FeynmanKac.IBoundaries

interface FeynmanKac.IInitialData

interface FeynmanKac.IForcingTerm

Usage Notes

Ordinary Differential Equations

An ordinary differential equation is an equation involving one or more dependent variables called $y_i$, one independent variable, $t$, and derivatives of the $y_i$ with respect to $t$.

In the initial-value problem (IVP), the initial or starting values of the dependent variables $y_i$ at a known value $t = t_0$ are given. Values of $y_i(t)$ for $t > 0$ or $t < t_0$ are required.

The OdeRungeKutta class solves the IVP for ODEs of the form

$$\frac{dy_i}{dt} = y'_i = f_i(t, y_1, ..., y_N) \quad i = 1, ..., N$$

with $y_i = (t = t_0)$ specified. Here, $f_i$ is a user-supplied function that must be evaluated at any set of
values \((t,y_1,\ldots,y_N), i = 1,\ldots,N\).

This problem statement is abbreviated by writing it as a system of first-order ODEs,

\[
y(t)\begin{bmatrix} y_1(t), \ldots, y_N(t) \\ f_1(t,y), \ldots, f_N(t,y) \end{bmatrix}^T
\]

so that the problem becomes \(y' = f(t,y)\) with initial values \(y(t_0)\).

The system

\[
\frac{dy}{dt} = y' = f(t,y)
\]

is said to be stiff if some of the eigenvalues of the Jacobian matrix

\[
\left\{ \frac{\partial y'}{\partial y_j} \right\}
\]

are large and negative. This is frequently the case for differential equations modeling the behavior of physical systems, such as chemical reactions proceeding to equilibrium where subspecies effectively complete their reactions in different epochs. An alternate model concerns discharging capacitors such that different parts of the system have widely varying decay rates (or time constants).

Users typically identify stiff systems by the fact that numerical differential equation solvers such as OdeRungeKutta are inefficient, or else completely fail. Special methods are often required. The most common inefficiency is that a large number of evaluations of \(f(t,y)\) (and hence an excessive amount of computer time) are required to satisfy the accuracy and stability requirements of the software.

### Partial Differential Equations

The FeynmanKac class solves the Feynman-Kac equation, a single partial differential equation, on a finite interval \([x_{\text{min}}, x_{\text{max}}]\). This equation often arises in applications from financial engineering and that is the primary focus of the documentation examples. The equation, initial conditions and Feynman-Kac boundary values are given by

\[
f_t + \mu(x,t) + \frac{\sigma^2(x,t)}{2}f_{xx} - \kappa(x,t)f = \phi(f,x,t),
\]

\[
f(x,T) = p(x), \{f_t = \frac{\partial f}{\partial t}, \text{etc.}\},
\]

\[
a(x,t)f + b(x,t)f_x + c(x,t)f_{xx} = d(x,t), \; x = x_{\text{min}} \text{ or } x = x_{\text{max}}.
\]

The solution is approximated by a piece-wise series of Hermite quintic polynomials on a grid of the interval \([x_{\text{min}}, x_{\text{max}}]\) that yields a twice differentiable solution.

To assist in the evaluation of the approximate solution and its derivatives there is method GetSplineValue.
ODE Class

public class Imsl.Math.ODE

ODE represents and solves an initial-value problem for ordinary differential equations.

Properties

Floor
virtual public double Floor {get; set; }

Description
The value used in the norm computation.

Property Value
A double used in the norm computation.
Default: Floor = 1.0.

Remarks
Floor must be greater than zero.

InitialStepsize
virtual public double InitialStepsize {get; set; }

Description
The initial internal step size.

Property Value
A double specifying the initial internal step size.
Default: InitialStepsize = 0.0.

Remarks
InitialStepsize must be greater than or equal to zero.

MaxSteps
virtual public int MaxSteps {get; set; }

Description
The maximum number of internal steps allowed.

Property Value
An int specifying the maximum number of internal steps allowed.
Default: MaxSteps = 500.
Remarks
MaxSteps must be greater than zero.

**MinimumStepsize**
virtual public double MinimumStepsize {get; set; }

Description
The minimum internal step size.

Property Value
A double specifying the minimum internal step size.
Default: MinimumStepsize = 0.0.

Remarks
MinimumStepsize must be greater than or equal to zero.

**NormMethod**
virtual public Imsl.Math.ODE.ErrorNormOptions NormMethod {get; set; }

Description
The error norm.

Property Value
An ODE.ErrorNormOptions specifying the error norm.
Default: NormMethod = ODE.ErrorNormOptions.MinAbsRel.

Remarks
NormMethod must be one of the values specified in the table which follows. In the following table, $e_i$ is the absolute value for an estimate of the error in $y_i(t)$.

<table>
<thead>
<tr>
<th>value</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinAbsRel</td>
<td>Minimum of the absolute error and the relative error, equals the maximum of $e_i / \max(</td>
</tr>
<tr>
<td>Abs</td>
<td>Absolute error, equals $\max(e_i)$</td>
</tr>
<tr>
<td>Max</td>
<td>Maximum of $e_i / \max(</td>
</tr>
<tr>
<td>Euclidean</td>
<td>Scaled Euclidean norm defined as $s = \sqrt{\sum_{i=1}^{\text{neq}} \frac{e_i^2}{w_i^2}}$</td>
</tr>
</tbody>
</table>

where $w_i = e_i / \max(|y_i(t)|, 1.0)$ and $\text{neq}$ is the number of equations

**Scale**
virtual public double Scale {get; set; }

Imsl C# Numerical Library
**Description**
The scaling factor.

**Property Value**
A double specifying the scaling factor.
Default: Scale = 1.0.

**Remarks**
Scale must be greater than zero.

---

**Tolerance**
virtual public double Tolerance {get; set; }

**Description**
The error tolerance.

**Property Value**
A double specifying the error tolerance.
Default: Tolerance = 1.0e-6.

**Remarks**
Tolerance must be greater than zero.

---

**Constructor**

**ODE**

ODE()

**Description**
Initializes a new instance of the Imsl.Math.ODE (p. 227) class.

---

**Methods**

**ExamineStep**

virtual void ExamineStep(Imsl.Math.ODE.ExamineStepOptions state, double t, double[] y)

**Description**
Called before and after each internal step.
Parameters

- state – An ODE.ExamineStepOptions, one of BeforeStep, AfterSuccessfulStep or AfterUnsuccessfulStep.
- t – A double representing the independent variable.
- y – A double array containing the dependent variables.

Remarks
This method can be over-ridden by the user to examine intermediate values of t and y.

GetMaximumStepsize

virtual public double GetMaximumStepsize()

Description
Returns the maximum internal step size.

Returns
A double specifying the maximum internal step size.

SetMaximumStepsize

virtual public void SetMaximumStepsize(double stepsize)

Description
Sets the maximum internal step size.

Parameter
- stepsize – A double specifying the maximum internal step size. stepsize must be greater than zero.

Remarks
Default: See SetMaximumStepsize in the subclasses for the default values used.

Vnorm

virtual double Vnorm(double[] v, double[] y, double[] ymax)

Description
Returns the norm of a vector.

Parameters
- v – A double array containing the vector whose norm is to be computed.
- y – A double array containing the values of the dependent variable.
- ymax – A double array containing the maximum y values computed thus far.

Returns
A double scalar value representing the norm of the vector v.
Remarks
This method can be over-ridden by the user to supply a different norm than those available through ODE.ErrorNormOptions, see NormMethod property.

ODE.ExamineStepOptions Enumeration

public enumeration Imsl.Math.ODE.ExamineStepOptions
ExamineStep options

Fields

AfterSuccessfulStep
public Imsl.Math.ODE.ExamineStepOptions AfterSuccessfulStep

Description
Indicates examining after a successful step.

AfterUnsuccessfulStep
public Imsl.Math.ODE.ExamineStepOptions AfterUnsuccessfulStep

Description
Indicates examining after an unsuccessful step.

BeforeStep
public Imsl.Math.ODE.ExamineStepOptions BeforeStep

Description
Indicates examining before the next step.

ODE.ErrorNormOptions Enumeration

public enumeration Imsl.Math.ODE.ErrorNormOptions
ErrorNorm options
Fields

Abs
public Imsl.Math.ODE.ErrorNormOptions Abs

Description
Indicates that the error norm to be used is to be the absolute error, equals \( \max(|e_i|) \).

Euclidean
public Imsl.Math.ODE.ErrorNormOptions Euclidean

Description
Indicates that the error norm to be used is to be the scaled Euclidean norm defined as

\[
s = \sqrt{\sum_{i=1}^{neq} \frac{e_i^2}{w_i^2}}
\]

where \( w_i = e_i / \max(|y_i(t)|, 1.0) \) and \( neq \) is the number of equations.

Max
public Imsl.Math.ODE.ErrorNormOptions Max

Description
Indicates that the error norm to be used is to be the maximum of \( e_i / \max(|y_i(t)|, \text{floor}) \) where \( \text{floor} \) is set via the Floor property.

MinAbsRel
public Imsl.Math.ODE.ErrorNormOptions MinAbsRel

Description
Indicates that the error norm to be used is to be the minimum of the absolute error and the relative error, equals the maximum of \( e_i / \max(|y_i(t)|, 1) \).

OdeRungeKutta Class

public class Imsl.Math.OdeRungeKutta : ODE

Solves an initial-value problem for ordinary differential equations using the Runge-Kutta-Verner fifth-order and sixth-order method.

Class OdeRungeKutta finds an approximation to the solution of a system of first-order differential equations of the form \( \frac{dy}{dt} = y' = f(t,y) \) with given initial data. The class attempts to keep the global error proportional to a user-specified tolerance. This class is efficient for nonstiff systems where the derivative evaluations are not expensive.
OdeRungeKutta is based on a code designed by Hull, Enright and Jackson (1976, 1977). It uses Runge-Kutta formulas of order five and six developed by J. H. Verner.

**Constructor**

**OdeRungeKutta**

```java
public OdeRungeKutta(Imsl.Math.OdeRungeKutta.IFunction function)
```

**Description**

Constructs an ODE solver to solve the initial value problem \( \frac{dy}{dt} = f(t,y) \).

**Parameter**

- `function` – Implementation of interface IFunction that defines the right-hand side function \( f(t,y) \).

**Methods**

**SetMaximumStepsize**

```java
override public void SetMaximumStepsize(double stepsize)
```

**Description**

Sets the maximum internal step size.

**Parameter**

- `stepsize` – A double specifying the maximum internal step size. `stepsize` must be greater than zero.
  
  Default: `stepsize = 2`.

**Solve**

```java
virtual public void Solve(double t, double tEnd, double[] y)
```

**Description**

Integrates the ODE system from \( t \) to \( tEnd \).

**Parameters**

- `t` – A double specifying the independent variable.
- `tEnd` – A double specifying the value of \( t \) at which the solution is desired.
- `y` – On input, a double array containing the initial values. On output, a double array containing the approximate solution.

**Remarks**

On all but the first call to `Solve`, the value of \( t \) must equal the value of \( tEnd \) from the previous call.
Exceptions

Imsl.Math.DidNotConvergeException is thrown if the number of internal steps exceeds MaxSteps (default 500). This can be an indication that the ODE system is stiff. This exception can also be thrown if the error tolerance condition could not be met.

Imsl.Math.ToleranceTooSmallException is thrown if the computation does not converge on some step.

Example: Runge-Kutta-Verner ordinary differential equation solver

An ordinary differential equation problem is solved using a solver which implements the Runge-Kutta-Verner method. The solution at time t=10 is printed.

```csharp
using System;
using Imsl.Math;

public class OdeRungeKuttaEx1 : OdeRungeKutta.IFunction
{
    public void F(double t, double[] y, double[] yprime)
    {
        yprime[0] = 2.0 * y[0] * (1 - y[1]);
        yprime[1] = -y[1] * (1 - y[0]);
    }

    public static void Main(String[] args)
    {
        double[] y = new double[]{1, 3};
        OdeRungeKutta q = new OdeRungeKutta(new OdeRungeKuttaEx1());
        int nsteps = 10;
        for (int k = 0; k < nsteps; k++)
        {
            q.Solve(k, k + 1, y);
        }
    }
}
```

Output

```
Result = {3.14434167651608,0.3488265985197}
```

OdeRungeKutta.IFunction Interface

public interface Imsl.Math.OdeRungeKutta.IFunction

Public interface for user supplied function to OdeRungeKutta object.
Method

F
abstract public void F(double t, double[] y, double[] yprime)

Description
Returns the value of the function at the given point.

Parameters
  t – A double, the point at which the function is to be evaluated.
  y – A double array which contains the dependent variable values.
  yprime – A double array which contains the value of the function at (t,y).

OdeAdamsGear Class

public class Imsl.Math.OdeAdamsGear : ODE

Extension of the ODE class to solve a stiff initial-value problem for ordinary differential equations using the Adams-Gear methods. Class OdeAdamsGear finds an approximation to the solution of a system of first-order differential equations of the form

\[ \frac{dy}{dt} = y' = f(t, y) \]

with given initial conditions for y at the starting value for t. The class attempts to keep the global error proportional to a user-specified tolerance. The proportionality depends on the differential equation and the range of integration.

The code is based on using backward difference formulas not exceeding order five as outlined in Gear (1971) and implemented by Hindmarsh (1974). There is an optional use of the code that employs implicit Adams formulas. This use is intended for nonstiff problems with expensive functions \( y' = f(t, y) \).

Properties

IntegrationMethod
virtual public Imsl.Math.OdeAdamsGear.IntegrationType IntegrationMethod {get; set; }

Description
The integration method.
Property Value
An OdeAdamsGear.IntegrationType specifying the integration method to be used.
Default: IntegrationMethod = OdeAdamsGear.IntegrationType.BDF.

Remarks
IntegrationMethod must be one of the values specified in the table which follows.

<table>
<thead>
<tr>
<th>value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams</td>
<td>Use the implicit Adams method.</td>
</tr>
<tr>
<td>BDF</td>
<td>Use backward differentiation formula (BDF) methods.</td>
</tr>
</tbody>
</table>

MaximumFunctionEvaluations
virtual public int MaximumFunctionEvaluations {get; set; }

Description
The maximum number of function evaluations of y' allowed.

Property Value
An int specifying the maximum number of function evaluations of y' allowed.
Default: No limit is enforced.

Remarks
MaximumFunctionEvaluations must be greater than zero.

MaxOrder
virtual public int MaxOrder {get; set; }

Description
The highest order formula to use of implicit Adams type or BDF type.

Property Value
An int specifying the highest order formula to use of implicit Adams type or BDF type.
Default: MaxOrder = 12 for Adams and MaxOrder = 5 for BDF.

Remarks
MaxOrder must be greater than zero.

NumberOfFcnEvals
virtual public int NumberOfFcnEvals {get; }

Description
Returns the number of function evaluations of y' made.

NumberOfJacobianEvals
virtual public int NumberOfJacobianEvals {get; }
**Description**

Returns the number of Jacobian matrix evaluations used.

**Property Value**

An `int` specifying the number of Jacobian matrix evaluations used.

**NumberOfSteps**

```csharp
virtual public int NumberOfSteps {get; }
```

**Description**

Returns the number of internal steps taken.

**Property Value**

An `int` specifying the number of internal steps taken.

**SolveMethod**

```csharp
virtual public Imsl.Math.OdeAdamsGear.SolveOption SolveMethod {get; set; }
```

**Description**

The method for solving the formula equations.

**Property Value**


**Remarks**

Note that if the problem is stiff and a chord or modified Newton method is most efficient, use `ChordUserJacobian` or `ChordComputedJacobian`. `SolveMethod` must be one of the values specified in the table which follows.

<table>
<thead>
<tr>
<th>value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FunctionIteration</td>
<td>Use a function iteration or successive substitution method.</td>
</tr>
<tr>
<td>ChordUserJacobian</td>
<td>Use a chord or modified Newton method and a user-supplied Jacobian.</td>
</tr>
<tr>
<td>ChordComputedJacobian</td>
<td>Use a chord or modified Newton method and a divided differences Jacobian.</td>
</tr>
<tr>
<td>ChordComputedDiagonal</td>
<td>Use a chord method and a diagonal matrix based on a directional directive.</td>
</tr>
</tbody>
</table>

**Constructor**

```csharp
public OdeAdamsGear(Imsl.Math.OdeAdamsGear.IFunction function)
```

**Differential Equations**

*OdeAdamsGear • 237*
Description
Constructs an ODE solver to solve the initial value problem \( \frac{dy}{dt} = f(t,y) \).

Parameter
function – Implementation of interface IFunction that defines the right-hand side function. \( f(t,y) \)

Methods

SetMaximumStepsize
override public void SetMaximumStepsize(double stepsize)

Description
Sets the maximum internal step size.

Parameter
stepsize – A double specifying the maximum internal step size. stepsize must be greater than zero.

Remarks
Default: stepsize = 1.7976931348623157e+308.

Solve
virtual public void Solve(double t, double tEnd, double[] y)

Description
Integrates the ODE system from \( t \) to \( tEnd \).

Parameters
t – A double specifying the independent variable.
tEnd – A double specifying the value of \( t \) at which the solution is desired.
y – On input, a double array containing the initial values. On output, a double array containing the approximate solution.

Remarks
On all but the first call to Solve, the value of \( t \) must equal the value of \( tEnd \) from the previous call.

Exceptions
Imsl.Math.DidNotConvergeException is thrown if the number of internal steps exceeds MaxSteps (default 500). This can be an indication that the ODE system is stiff. This exception can also be thrown if the error tolerance condition could not be met.
Imsl.Math.ToleranceTooSmallException is thrown if the computation does not converge on some step.
Imsl.Math.MaxFcnEvalsExceeded is thrown if the maximum number of function evaluations allowed has been exceeded.
Imsl.Math.SingularMatrixException is thrown if the factorization function encounters a singular matrix during LU decomposition.
Example: Adams-Gear ordinary differential equation solver

A mildly stiff ordinary differential equation problem is solved using a solver which implements the Adams-Gear method. The solution at time \( t=240 \) is printed.

```csharp
using System;
using Imsl.Math;

public class OdeAdamsGearEx1
{
    private class MyFunction : OdeAdamsGear.IFunction
    {
        public MyFunction(double k1, double k2, double k3)
        {
            this.k1 = k1;
            this.k2 = k2;
            this.k3 = k3;
        }

        private double k1;
        private double k2;
        private double k3;

        public virtual double[] F(double t, double[] y)
        {
            double[] yprime = new double[y.Length];
            yprime[0] = - y[0] - y[0] * y[1] + k1 * y[1];
            yprime[1] = (- k2) * y[1] + k3 * (1.0 - y[1]) * y[0];
            return yprime;
        }
    }

    public static void Main(String[] args)
    {
        double k1 = 294.0;
        double k2 = 3.0;
        double k3 = 0.01020408;
        OdeAdamsGear.IFunction f = new MyFunction(k1, k2, k3);
        double t = 0.0;
        double tend = 240.0;
        double[] y = {1.0, 0.0};
        OdeAdamsGear q = new OdeAdamsGear(f);
        q.NormMethod = OdeAdamsGear.ErrorNormOptions.Abs;
        q.SolveMethod = OdeAdamsGear.SolveOption.ChordComputedJacobian;
        q.Tolerance = 1e-3;
        q.Solve(t, tend, y);
        // Print Results
        Console.Out.WriteLine("Result = {{{0,5:0.####}, {1,5:0.####}}}, y[0], y[1]};
    }
}
```
Output
Result = {0.3924, 0.0013}

OdeAdamsGear.IntegrationType Enumeration

public enumeration Imsl.Math.OdeAdamsGear.IntegrationType
Integration type.

Fields

Adams
public Imsl.Math.OdeAdamsGear.IntegrationType Adams
Description
Use the implicit Adams method.

BDF
public Imsl.Math.OdeAdamsGear.IntegrationType BDF
Description
Use backward differentiation formula (BDF) methods.

OdeAdamsGear.SolveOption Enumeration

public enumeration Imsl.Math.OdeAdamsGear.SolveOption
Solve option.

Fields

ChordComputedDiagonal
public Imsl.Math.OdeAdamsGear.SolveOption ChordComputedDiagonal
Description
Use a chord method and a diagonal matrix based on a directional directive.

ChordComputedJacobian

Description
Use a chord or modified Newton method and a Jacobian approximated by divided differences.

ChordUserJacobian

Description
Use a chord or modified Newton method and a user-supplied Jacobian.

FunctionIteration
public Imsl.Math.OdeAdamsGear.SolveOption FunctionIteration

Description
Use a function iteration or successive substitution method.

OdeAdamsGear.IFunction Interface

public interface Imsl.Math.OdeAdamsGear.IFunction
Public interface for user supplied function to OdeAdamsGear object.

Method

F
abstract public double[] F(double t, double[] y)

Description
Computes the value of the function \( y' = f(t,y) \) at the given point.

Parameters
- \( t \) – A double, the point at which the function is to be evaluated.
- \( y \) – A double array which contains the dependent variable values.
Returns
A double array of length \( y.\text{Length} \) which contains the value of the function
\[
\frac{dy}{dt} = y' = f(t,y).
\]

---

**OdeAdamsGear.IJacobian Interface**

```csharp
public interface Imsl.Math.OdeAdamsGear.IJacobian :
    Imsl.Math.OdeAdamsGear.IFunction
{]
Public interface for the user supplied function to evaluate the Jacobian matrix.

**Method**

**Jacobian**

```csharp
abstract public double[,] Jacobian(double t, double[] y, double[] yprime)
```

**Description**

Used to compute the Jacobian of the function at \( t \).

**Parameters**

- \( t \) – A double, the point at which the function is to be evaluated.
- \( y \) – A double array which contains the dependent variable values.
- \( yprime \) – A double array which contains the value of the function \( \frac{dy}{dt} = y' = f(t,y) \).

**Returns**

A double \( y.\text{Length} \) by \( y.\text{Length} \) matrix containing the value of the Jacobian of the function at \( t \).

---

**FeynmanKac Class**

```csharp
public class Imsl.Math.FeynmanKac
{]
Solves the generalized Feynman-Kac PDE.

Class FeynmanKac solves the generalized Feynman-Kac PDE on a rectangular grid with boundary
conditions using a finite element Galerkin method. The generalized Feynman-Kac differential equation
has the form
\[
f_t + \mu(x,t)f_x + \frac{\sigma^2(x,t)}{2}f_{xx} - \kappa(x,t)f = \phi(f,x,t),
\]
where the initial data satisfies \( f(x, T) = p(x) \). The derivatives are \( f_i = \frac{df}{dx} \), etc. Method ComputeCoefficients uses a finite element Galerkin method over the rectangle
\[
[x_{\min}, x_{\max}] \times [T, T]
\]
in \((x, t)\) to compute the approximate solution. The interval \([x_{\min}, x_{\max}]\) is decomposed with a grid
\[
(x_{\min} = )x_1 < x_2 < \ldots < x_m (= x_{\max}).
\]
On each subinterval the solution is represented by
\[
f(x, t) = f_i b_0(z) + f_{i+1} b_0(1 - z) + h_i f_i' b_1(z) - h_i f_{i+1}' b_1(1 - z) + h_i^2 f_i'' b_2(z) + h_i^2 f_{i+1}'' b_2(1 - z).
\]
The values
\[
f_i, f_i', f_i'', f_{i+1}, f_{i+1}', f_{i+1}''
\]
are time-dependent coefficients associated with each interval. The basis functions \( b_0, b_1, b_2 \) are given for
\[x \in [x_i, x_{i+1}], \quad h_i = x_{i+1} - x_i, \quad z = (x - x_i)/h_i, \quad z \in [0, 1],\]
by
\[
\begin{align*}
b_0(z) &= -6z^5 + 15z^4 - 10z^3 + 1 = (1 - z)^3(6z^2 + 3z + 1), \\
b_1(z) &= -3z^5 + 8z^4 - 6z^3 + z = (1 - z)^3(3z + 1), \\
b_2(z) &= \frac{1}{2}(-z^5 + 3z^4 - 3z^3 + z^2) = \frac{1}{2}(1 - z)^3 z^2.
\end{align*}
\]
By adding the piece-wise definitions the unknown solution function may be arranged as the series
\[f(x, t) = \sum_{i=1}^{3m} y_i \beta_i(x), \quad x \in [x_{\min}, x_{\max}],\]
where the time-dependent coefficients are defined by re-labeling:
\[y_{3i-2} = f_i, \quad y_{3i-1} = f_i', \quad y_{3i} = f_i'', \quad i = 1, \ldots, m.\]
The Galerkin principle is then applied. Using the provided initial and boundary conditions leads to an
Index 1 differential-algebraic equation for the coefficients \( y_i, i = 1, \ldots, 3m. \)

This system is integrated using the variable order, variable step algorithm DDASLX noted in Hanson, R. and Krogh, F. (2008), *Solving Constrained Differential-Algebraic Systems Using Projections*. Solution values and their time derivatives at a grid preceding time \( T \), expressed in units of time remaining, are given back by methods GetSplineCoefficients and GetSplineCoefficientsPrime, respectively. For further details of deriving and solving the system see Hanson, R. (2008), *Integrating Feynman-Kac Equations Using Hermite Quintic Finite Elements*. To evaluate \( f \) or its partials
\[f_x, f_{xx}, f_{xxx}, f_{t}, f_{tx}, f_{txx}, f_{txxx}\]
at any time point in the grid, use method GetSplineValue.

One useful application of the FeynmanKac class is financial analytics. This is illustrated in Example 2, which solves a diffusion model for call options which, in the special case \( \alpha = 2 \) reduces to the Black-Scholes (BS) model. Another useful application for the FeynmanKac class is the calculation of the Greeks, i.e. various derivatives of Feynman-Kac solutions applicable to, e.g., the pricing of options.
and related financial derivatives. In Example 5, the FeynmanKac class is used to calculate eleven of the Greeks for the same diffusion model introduced in Example 2 in the special BS case. These Greeks are also calculated using the BS closed form Greek equations (see http://en.wikipedia.org/wiki/The_Greeks). The Feynman-Kac and BS solutions are output and compared. Example 5 illustrates that the FeynmanKac class can be used to explore the Greeks for a much wider class of financial models than can BS.

Properties

**GaussLegendreDegree**

```csharp
public int GaussLegendreDegree { get; set; }
```

**Description**
The number of quadrature points used in the Gauss-Legendre quadrature formula.

**Property Value**
An int scalar, the degree of the polynomial used in the Gauss-Legendre quadrature.
Default: GaussLegendreDegree = 6.

**Remarks**
It is required that the degree of the polynomial is greater than or equal to 6.

**InitialStepsize**

```csharp
public double InitialStepsize { get; set; }
```

**Description**
The starting step size for the integration.

**Property Value**
A double scalar, the starting step size used in the integrator.
Default: InitialStepsize = -1.1102230246252e-16.

**Remarks**
The starting step size must be less than zero since the integration is internally done from \( t=0 \) to \( t=t\text{Grid}[t\text{Grid.Length}-1] \) in a negative direction.

**MaximumBDFOrder**

```csharp
public int MaximumBDFOrder { get; set; }
```

**Description**
The maximum order of the backward differentiation formulas (BDF).

**Property Value**
An int, the maximum order of the BDF used in the integrator.
Default: MaximumBdfOrder = 5.
Remarks
The maximum order of the BDF must be greater than zero and less than 6.

MaximumStepsize
public double MaximumStepsize {get; set; }

Description
The maximum internal step size used by the integrator.

Property Value
A positive scalar of type double, the maximum internal step size.
Default: MaximumStepsize = Double.MaxValue, the largest possible machine number.

MaxSteps
public int MaxSteps {get; set; }

Description
The maximum number of internal steps allowed.

Property Value
An int scalar specifying the maximum number of internal steps allowed between each output point of the integration.
Default: MaxSteps = 500000.

Method
public Imsl.Math.FeynmanKac.PDEStepControlMethod Method {get; set; }

Description
The step control method used in the integration of the Feynman-Kac PDE.

Property Value
A PDEStepControlMethod value specifying the step control method used in the integration.

Remarks

<table>
<thead>
<tr>
<th>PDEStepControlMethod</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MethodOfSoederlind</td>
<td>Use method of Soederlind.</td>
</tr>
<tr>
<td>MethodOfPetzold</td>
<td>Use method from the original Petzold code DASSL.</td>
</tr>
</tbody>
</table>

TimeBarrier
public double TimeBarrier {get; set; }

Description
Sets or returns the barrier used for integration in the time direction.
**Property Value**
A double scalar, the time point beyond which the integrator should not integrate.
Default: `TimeBarrier = tGrid[tGrid.Length-1]`.

**Remarks**
This time barrier controls whether the integrator should integrate in the time direction beyond a special point, `TimeBarrier`, and then interpolate to get the Hermite quintic spline coefficients and its derivatives at the points in `tGrid`. It is required that `TimeBarrier` be greater than or equal to `tGrid[tGrid.Length-1]`.

**Constructor**

**FeynmanKac**
```csharp
public FeynmanKac(Imsl.Math.FeynmanKac.IPdeCoefficients pdeCoeffs)
```

**Description**
Constructs a PDE solver to solve the Feynman-Kac PDE.

**Parameter**
- `pdeCoeffs` – Implementation of interface `IPdeCoefficients` that computes the values of the Feynman-Kac coefficients at a given point `(t,x)`.

**Methods**

**ComputeCoefficients**
```csharp
public void ComputeCoefficients(int numLeftBounds, int numRightBounds,
Imsl.Math.FeynmanKac.IBoundaries pdeBounds, double[] xGrid, double[] tGrid)
```

**Description**
Determines the coefficients of the Hermite quintic splines that represent an approximate solution for the Feynman-Kac PDE.

**Parameters**
- `numLeftBounds` – An int scalar, the number of left boundary conditions. It is required that `1 ≤ numLeftBounds ≤ 3`.
- `numRightBounds` – An int scalar, the number of right boundary conditions. It is required that `1 ≤ numRightBounds ≤ 3`.
- `pdeBounds` – Implementation of interface `IBoundaries` that computes the boundary coefficients and terminal condition for given `(t,x)`.
- `xGrid` – A double array containing the breakpoints for the Hermite quintic splines used in the `x` discretization. The length of `xGrid` must be at least 2, `xGrid.Length ≥ 2`, and the elements in `xGrid` must be in strictly increasing order.
tGrid – A double array containing the set of time points (in time-remaining units) where an approximate solution is returned. The elements in array tGrid must be positive and in strictly increasing order.

Exceptions

- **Imsl.Math.ToleranceTooSmallException** is thrown if the absolute or relative error tolerances used in the integrator are too small.
- **Imsl.Math.TooManyStepsException** is thrown if the integrator needs too many iteration steps.
- **Imsl.Math.ErrorTestException** is thrown if the error test used in the integrator failed repeatedly.
- **Imsl.Math.CorrectorConvergenceException** is thrown if the corrector failed to converge repeatedly.
- **Imsl.Math.IterationMatrixSingularException** is thrown if one of the iteration matrices used in the integrator is singular.
- **Imsl.Math.TimeIntervalTooSmallException** is thrown if the distance between an intermediate starting and end point for the integration is too small.
- **Imsl.Math.TCurrentTStopInconsistentException** is thrown if during the integration the current integration time and given stepsize is inconsistent with the endpoint of the integration.
- **Imsl.Math.TEqualsTOutException** is thrown if during the integration process the actual integration time and the end time of the integration are identical.
- **Imsl.Math.InitialConstraintsException** is thrown if at the initial integration point some of the constraints are inconsistent.
- **Imsl.Math.ConstraintsInconsistentException** is thrown if during the integration process the constraints for the actual time point and given stepsize are inconsistent.
- **Imsl.Math.SingularMatrixException** is thrown if one of the matrices used outside the integrator is singular.
- **Imsl.Math.BoundaryInconsistentException** is thrown if the boundary conditions are inconsistent.

**GetAbsoluteErrorTolerances**

```csharp
public double[] GetAbsoluteErrorTolerances()
```

**Description**

Returns absolute error tolerances.

**Returns**

A double array of length 3*xGrid.Length containing absolute error tolerances for the solutions.

**GetRelativeErrorTolerances**

```csharp
public double[] GetRelativeErrorTolerances()
```

**Description**

Returns relative error tolerances.
Returns
A double array of length 3*xGrid.Length containing relative error tolerances for the solutions.

GetSplineCoefficients
public double[,] GetSplineCoefficients()

Description
Returns the coefficients of the Hermite quintic splines that represent an approximate solution of the Feynman-Kac PDE.

Remarks
The ComputeCoefficients method must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.

GetSplineCoefficientsPrime
public double[,] GetSplineCoefficientsPrime()

Description
Returns the first derivatives of the Hermite quintic spline coefficients that represent an approximate solution of the Feynman-Kac PDE.

Remarks
The ComputeCoefficients method must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.
and

\[ f_t(x,\bar{t}) = \sum_{j=0}^{3*nxGrid-1} y'_0^j \beta_j(x) \quad \text{for } \bar{t} = 0. \]

The \((ntGrid+1)\) by \((3*nxGrid)\) matrix

\[
(y'_i^j)_{j=0,...,3*nxGrid-1}^{i=0,...,ntGrid}
\]

is stored row-wise in the returned array.

**Remarks**

The `ComputeCoefficients` method must be invoked first before invoking this method. Otherwise, the method throws an `InvalidOperationException` exception.

**GetSplineValue**

```csharp
public double[] GetSplineValue(double[] evaluationPoints, double[] coefficients, int ideriv)
```

**Description**

Evaluates for time value 0 or a time value in `tGrid` the derivative of the Hermite quintic spline interpolant at evaluation points within the range of `xGrid`.

**Parameters**

- `evaluationPoints` – A double array containing the points in x-direction at which the Hermite quintic spline representing the approximate solution to the Feynman-Kac PDE or one of its derivatives is to be evaluated. It is required that all elements in array `evaluationPoints` are greater than or equal to `xGrid[0]` and less than or equal to `xGrid[xGrid.Length-1].`

- `coefficients` – A `double` array of length \(3*\) `xGrid.Length` containing the coefficients of the Hermite quintic spline representing the approximate solution \(f\) or \(f_t\) to the Feynman-Kac PDE. These coefficients are the rows of the arrays `splineCoeffs` and `splineCoeffsPrime` returned by methods `GetSplineCoefficients` and `GetSplineCoefficientsPrime`. If the user wants to compute approximate solutions \(f\) or \(f_t, f_{xx}, f_{xxx}\) to the Feynman-Kac PDE at time point 0, one must assign row 0 of `splineCoeffs` to array `coefficients`. If the user wants to compute these approximate solutions for time points \(t=tGrid[i], i=0,...,tGrid.Length-1\), one must assign row \(i+1\) of `splineCoeffs` to array `coefficients`. The same reasoning applies to the computation of approximate solutions \(f_t\) and \(f_{xx}, f_{xxx}\) and assignment of rows of array `splineCoeffsPrime` to array `coefficients`.

- `ideriv` – An `int` specifying the derivative to be computed. It must be 0, 1, 2 or 3.

**Returns**

A `double` array containing the derivative of order `ideriv` of the Hermite quintic spline representing the approximate solution \(f\) or \(f_t\) to the Feynman Kac PDE at `evaluationPoints`. If `ideriv`=0, then the spline values are returned. If `ideriv`=1, then the first derivative is returned, etc.

**SetAbsoluteErrorTolerances**

```csharp
public void SetAbsoluteErrorTolerances(double[] atol)
```
Description

Sets the absolute error tolerances.

Parameter

\texttt{atol} – A double array of length 3*\texttt{xGrid.Length} specifying the absolute error tolerances for the row-wise solutions returned by method \texttt{GetSplineCoefficients}. All entries in \texttt{atol} must be greater than or equal to zero. Also, not all entries in \texttt{atol} and \texttt{rtol} are allowed to be equal to 0 simultaneously.

Default: \texttt{atol[i] = 1.0e-5} for \texttt{i=0,...,3*xGrid.Length-1}.

\textbf{SetAbsoluteErrorTolerances}

public void SetAbsoluteErrorTolerances(double atol)

Description

Sets the absolute error tolerances.

Parameter

\texttt{atol} – A double scalar specifying the absolute error tolerances for the row-wise solutions returned by method \texttt{GetSplineCoefficients}. The tolerance value \texttt{atol} is applied to all 3*\texttt{xGrid.Length} solution components. \texttt{atol} must be greater than or equal to zero. Also, not all entries in \texttt{atol} and \texttt{rtol} are allowed to be equal to 0 simultaneously.

Default: \texttt{atol} = 1.0e-5.

\textbf{SetForcingTerm}

public void SetForcingTerm(Imsl.Math.FeynmanKac.IForcingTerm forceTerm)

Description

Sets the user-supplied method that computes approximations to the forcing term \( \phi(x) \) and its derivative \( \frac{\partial \phi}{\partial y} \) used in the FeynmanKac PDE.

Parameter

\texttt{forceTerm} – An \texttt{IForcingTerm} object specifying the user defined function used for computation of the forcing term \( \phi(f,x,t) \) and its derivative \( \frac{\partial \phi}{\partial y} \).

Default: If this member function is not called it is assumed that \( \phi(f,x,t) \) is identically zero.

\textbf{SetInitialData}

public void SetInitialData(Imsl.Math.FeynmanKac.IInitialData initData)

Description

Sets the user-supplied method for adjustment of initial data or as an opportunity for output during the integration steps.

Parameter

\texttt{initData} – An \texttt{IInitialData} object specifying the user-defined function for adjustment of initial data or as an opportunity for output during the integration steps.

Default: No adjustment of initial data or output during the integration steps is done.

\textbf{SetRelativeErrorTolerances}

public void SetRelativeErrorTolerances(double[] rtol)
Description
Sets the relative error tolerances.

Parameter
- **rtol** – A double array of length 3*xGrid.Length specifying the relative error tolerances for the row-wise solutions returned by method GetSplineCoefficients. All entries in rtol must be greater than or equal to zero. Also, not all entries in atol and rtol are allowed to be equal to 0 simultaneously.

   Default: rtol[i] = 1.0e-5 for i=0,...,3*xGrid.Length-1.

**SetRelativeErrorTolerances**
public void SetRelativeErrorTolerances(double rtol)

Description
Sets the relative error tolerances.

Parameter
- **rtol** – A double scalar specifying the relative error tolerances for the row-wise solutions returned by method GetSplineCoefficients. The tolerance value rtol is applied to all 3*xGrid.Length solution components. rtol must be greater than or equal to zero. Also, not all entries in atol and rtol are allowed to be equal to 0 simultaneously.

   Default: rtol = 1.0e-5.

**SetTimeDependence**
public void SetTimeDependence(bool[] timeFlag)

Description
Sets the time dependence of the coefficients, boundary conditions and function φ in the Feynman-Kac equation.

Parameter
- **timeFlag** – A bool vector of length 7 indicating time dependencies in the Feynman-Kac PDE.

<table>
<thead>
<tr>
<th>Index</th>
<th>Time dependency of</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>σ'</td>
</tr>
<tr>
<td>1</td>
<td>σ</td>
</tr>
<tr>
<td>2</td>
<td>μ</td>
</tr>
<tr>
<td>3</td>
<td>κ</td>
</tr>
<tr>
<td>4</td>
<td>Left boundary conditions</td>
</tr>
<tr>
<td>5</td>
<td>Right boundary conditions</td>
</tr>
<tr>
<td>6</td>
<td>φ</td>
</tr>
</tbody>
</table>

Remarks
- *timeFlag[i]* = true indicates that the associated value is time-dependent, *timeFlag[i]* = false indicates that the associated value is time-independent.

   Default: *timeFlag[i]* = false for i = 0,...,6.
Example 1: American Option vs. European Option on a Vanilla Put

The value of the American Option on a Vanilla Put can be no smaller than its European counterpart. That is due to the American Option providing the opportunity to exercise at any time prior to expiration. This example compares this difference - or premium value of the American Option - at two time values using the Black-Scholes model. The example is based on Wilmott et al. (1996, p. 176), and uses the non-linear forcing or weighting term described in Hanson, R. (2008), Integrating Feynman-Kac Equations Using Hermite Quintic Finite Elements, for evaluating the price of the American Option. The coefficients, payoff, boundary conditions and forcing term for American and European options are defined through interfaces IPdeCoefficients, IBoundaries and IForcingTerm, respectively. One breakpoint is set exactly at the strike price. The sets of parameters in the computation are:

1. Strike price $K = 10.0$
2. Volatility $\sigma = 0.4$
3. Times until expiration = \{1/4, 1/2\}
4. Interest rate $r = 0.1$
5. $x_{\min} = 0.0$, $x_{\max} = 30.0$
6. $nxGrid = 61$, $n = 3 \times nxGrid = 183$

The payoff function is the “vanilla option”, $p(x) = \max(K - x, 0)$.

```csharp
using System;
using Imsl.Math;

public class FeynmanKacEx1 : FeynmanKac.IPdeCoefficients, FeynmanKac.IBoundaries
{
    // The coefficient sigma(x)
    public double Sigma(double x, double t)
    {
        double sigma = 0.4;
        return (sigma * x);
    }

    // The coefficient derivative d(sigma) / dx
    public double SigmaPrime(double x, double t)
    {
        double sigma = 0.4;
        return sigma;
    }

    // The coefficient mu(x)
    public double Mu(double x, double t)
    {
        double interestRate = 0.1;
        double dividend = 0.0;
```
return ((interestRate - dividend) * x);
}

// The coefficient kappa(x)
public double Kappa(double x, double t)
{
    double interestRate = 0.1;
    return interestRate;
}

public void LeftBoundaries(double time, double[,] bndCoeffs)
{
    bndCoeffs[0, 0] = 0.0;
    bndCoeffs[0, 1] = 1.0;
    bndCoeffs[0, 2] = 0.0;
    bndCoeffs[0, 3] = -1.0;
    bndCoeffs[1, 0] = 0.0;
    bndCoeffs[1, 1] = 0.0;
    bndCoeffs[1, 2] = 1.0;
    bndCoeffs[1, 3] = 0.0;
}

public void RightBoundaries(double time, double[,] bndCoeffs)
{
    bndCoeffs[0, 0] = 1.0;
    bndCoeffs[0, 1] = 0.0;
    bndCoeffs[0, 2] = 0.0;
    bndCoeffs[0, 3] = 0.0;
    bndCoeffs[1, 0] = 0.0;
    bndCoeffs[1, 1] = 1.0;
    bndCoeffs[1, 2] = 0.0;
    bndCoeffs[1, 3] = 0.0;
    bndCoeffs[2, 0] = 0.0;
    bndCoeffs[2, 1] = 0.0;
    bndCoeffs[2, 2] = 1.0;
    bndCoeffs[2, 3] = 0.0;
}

public double Terminal(double x)
{
    double zero = 0.0;
    // Strike price
    double strikePrice = 10.0;
    // The payoff function
    double val = Math.Max(strikePrice - x, zero);
    return val;
}

class MyForcingTerm : FeynmanKac.IForcingTerm
{
    public void Force(int interval, double[] y, double time, double width,
                       double[] xlocal, double[] qw, double[,] u, double[] phi,
                       double[,] dphi)
    {
        const int local = 6;
    
    class MyForcingTerm : FeynmanKac.IForcingTerm
    {
        public void Force(int interval, double[] y, double time, double width,
                           double[] xlocal, double[] qw, double[,] u, double[] phi,
                           double[,] dphi)
        {
            const int local = 6;
        }
const double zero = 0.0;
const double one = 1.0;
double[] yl = new double[local];
double[] bf = new double[local];
double val, strikePrice, interestRate;
double rt, mu;
int nxGrid = y.Length / 3;
int ndeg = xlocal.Length;

for (int i = 0; i < local; i++)
{
    yl[i] = y[3 * interval - 3 + i];
    phi[i] = zero;
}

strikePrice = 10.0;
interestRate = 0.1;
val = 1.0e-5;
mu = 2.0;

// This is the local definition of the forcing term
for (int j = 1; j <= local; j++)
{
    for (int l = 1; l <= ndeg; l++)
    {
        bf[0] = u[0, l - 1];
        bf[1] = u[1, l - 1];
        bf[2] = u[2, l - 1];
        bf[3] = u[6, l - 1];
        bf[4] = u[7, l - 1];
        bf[5] = u[8, l - 1];
        rt = 0.0;
        for (int k = 0; k < local; k++)
        {
            rt += yl[k] * bf[k];
        }
        rt = val / (rt + val - (strikePrice - xlocal[l - 1]));
    }
}

for (int i = 0; i < local; i++)
{
    phi[i] = (-phi[i]) * width * interestRate * strikePrice;
}

// This is the local derivative matrix for the forcing term
for (int i = 0; i < local; i++)
{
    for (int j = 0; j < local; j++)
    {
        dphi[i, j] = zero;
    }
}

for (int j = 1; j <= local; j++)
{
    for (int i = 1; i <= local; i++)
    {
        for (int l = 1; l <= ndeg; l++)
        {
public static void Main(String[] args)
{
    // Compute American Option Premium for Vanilla Put
    // The strike price
    double KS = 10.0;
    // The sigma value
    double sigma = 0.4;
    // Time values for the options
    int nt = 2;
    double[] tGrid = { 0.25, 0.5 };
    // Values of the underlying where evaluations are made
    double[] evaluationPoints = { 0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0 };

    // Value of the interest rate
    double r = 0.1;
    // Values of the min and max underlying values modeled
    double xMin = 0.0, xMax = 30.0;
    // Define parameters for the integration step.
    int nxGrid = 61;
    int nv = 9;
    int nint = nxGrid - 1, n = 3 * nxGrid;
    double[] xGrid = new double[nxGrid];
    double dx;
    int nbc = 2, nrbc = 3;
    // Define an equally-spaced grid of points for the underlying price
}
dx = (xMax - xMin) / ((double)nint);
xGrid[0] = xMin;
xGrid[nxGrid - 1] = xMax;
for (int i = 2; i <= nxGrid - 1; i++)
{
    xGrid[i - 1] = xGrid[i - 2] + dx;
}

FeynmanKacEx1 ex1 = new FeynmanKacEx1();

FeynmanKac european = new FeynmanKac(ex1);
FeynmanKac american = new FeynmanKac(ex1);

MyForcingTerm forceTerm = new MyForcingTerm();
american.SetForcingTerm(forceTerm);

european.ComputeCoefficients(nlbcd, nrbcd, ex1, xGrid, tGrid);
american.ComputeCoefficients(nlbcd, nrbcd, ex1, xGrid, tGrid);

// Evaluate solutions at vector of points evaluationPoints, at each
// time value prior to expiration.

double[,] europeanCoefficients = european.GetSplineCoefficients();
double[,] americanCoefficients = american.GetSplineCoefficients();

double[] europeanTimeCoeffs = new double[n];
double[] americanTimeCoeffs = new double[n];

double[,] splineValuesEuropean = new double[nt][];
double[,] splineValuesAmerican = new double[nt][];

for (int i = 0; i < nt; i++)
{
    // Extract spline coefficients for individual times
    for (int j = 0; j < n; j++)
    {
        europeanTimeCoeffs[j] = europeanCoefficients[i + 1, j];
        americanTimeCoeffs[j] = americanCoefficients[i + 1, j];
    }

    splineValuesEuropean[i] =
        european.GetSplineValue(evaluationPoints, europeanTimeCoeffs, 0);
    splineValuesAmerican[i] =
        american.GetSplineValue(evaluationPoints, americanTimeCoeffs, 0);
}

Console.Out.WriteLine();
Console.Out.WriteLine("American Option Premium for Vanilla Put, " +
"3 and 6 Months Prior to Expiry");
Console.Out.WriteLine(" Number of equally spaced spline" +
" knots:{0,4:d}" , nxGrid);
Console.Out.WriteLine(" Number of unknowns:{0,4:d}" , n);
Console.Out.WriteLine(" Strike={1,5:f2}, sigma={2,5:f2}, " +
"Interest Rate={2,5:f2}" , KS, sigma, r);

Console.Out.WriteLine();
Example 2: A diffusion model for Call Options

In Beckers (1980) there is a model for a Stochastic Differential Equation of option pricing. The idea is a “constant elasticity of variance diffusion (or CEV) class”

\[ dS = \mu S dt + \sigma S^{\alpha/2} dW, \quad 0 < \alpha < 2. \]

The Black-Scholes model is the limiting case \( \alpha \to 2 \). A numerical solution of this diffusion model yields the price of a call option. Various values of the strike price \( K \), time values, \( \sigma \) and power coefficient \( \alpha \) are used to evaluate the option price at values of the underlying price. The sets of parameters in the computation are:

1. power \( \alpha = 2.0, 1.0, 0.0 \)
2. strike price \( K = 15.0, 20.0, 25.0 \)
3. volatility \( \sigma = 0.2, 0.3, 0.4 \)
4. times until expiration = \{1/12, 4/12, 7/12\}
5. underlying prices = \{19.0, 20.0, 21.0\}
6. interest rate \( r = 0.05 \)
7. \( x_{\text{min}} = 0, x_{\text{max}} = 60 \)
8. \( nxGrid = 121, n = 3 \times nxGrid = 363 \)

With this model the Feynman-Kac differential equation is defined by identifying:

- \( x \): \( S \)
- \( \sigma(x,t) \): \( \sigma x^{\alpha/2}, \frac{\partial \sigma}{\partial t} = \frac{\alpha}{2} x^{\alpha/2-1} \)
- \( \mu(x,t) \): \( rx \)
- \( \kappa(x,t) \): \( r \)
- \( \phi(f,x,t) \equiv 0 \)

The payoff function is the “vanilla option”, \( p(x) = \max(x - K, 0) \).

```csharp
using System;
using Imsl.Math;

public class FeynmanKacEx2
{
    public static void Main(String[] args)
    {
        // Compute Constant Elasticity of Variance Model for Vanilla Call
        // The set of strike prices
        double[] strikePrices = { 15.0, 20.0, 25.0 }; // The set of sigma values
        double[] sigma = { 0.2, 0.3, 0.4 }; // The set of model diffusion powers
        double[] alpha = { 2.0, 1.0, 0.0 }; // Time values for the options
        int nt = 3;
        double[] tGrid = { 1.0 / 12.0, 4.0 / 12.0, 7.0 / 12.0 }; // Values of the underlying where evaluations are made
        double[] evaluationPoints = { 19.0, 20.0, 21.0 }; // Value of the interest rate and continuous dividend
        double r = 0.05, dividend = 0.0; // Values of the min and max underlying values modeled
        double xMin = 0.0, xMax = 60.0; // Define parameters for the integration step. */
        int nxGrid = 121;
        int ntGrid = 3;
        int nv = 3;
        int nint = nxGrid - 1, n = 3 * nxGrid;
        double[] xGrid = new double[nxGrid];
        double dx;
        // Number of left/right boundary conditions
        int nlbcd = 3, nrbcd = 3;
    }
}
```
// Time dependency
bool[] timeDependence = new bool[7];
double[] userData = new double[6];

// Define equally-spaced grid of points for the underlying price
// dx = (xMax - xMin) / ((double)nint);
xGrid[0] = xMin;
xGrid[nxGrid - 1] = xMax;
for (int i = 2; i <= nxGrid - 1; i++)
{
    xGrid[i - 1] = xGrid[i - 2] + dx;
}

Console.Out.WriteLine(" Constant Elasticity of Variance Model" +
    " for Vanilla Call");

for (int i1 = 1; i1 <= 3; i1++)
/* Loop over power */
{
    for (int i2 = 1; i2 <= 3; i2++)
/* Loop over volatility */
    {
        for (int i3 = 1; i3 <= 3; i3++)
/* Loop over strike price */
            {
                // Pass data through into evaluation routines.
                userData[0] = strikePrices[i3 - 1];
                userData[1] = xMax;
                userData[2] = sigma[i2 - 1];
                userData[3] = alpha[i1 - 1];
                userData[4] = r;
                userData[5] = dividend;

                FeynmanKac callOption =
                    new FeynmanKac(new MyCoefficients(userData));

                // Right boundary condition is time-dependent
                timeDependence[5] = true;
                callOption.SetTimeDependence(timeDependence);
                callOption.ComputeCoefficients(nlbcd, nrbcd,
                    new MyBoundaries(userData), xGrid, tGrid);

                double[,] optionCoefficients =
                    Differential Equations FeynmanKac • 259
callOption.GetSplineCoefficients();
double[] optionTimeCoeffs = new double[n];
double[][] splineValuesOption = new double[ntGrid][];

// Evaluate solution at vector evaluationPoints, at each time
// value prior to expiration.
for (int i = 0; i < ntGrid; i++)
{
    for (int j = 0; j < n; j++)
        optionTimeCoeffs[j] = optionCoefficients[i + 1, j];
    splineValuesOption[i] =
        callOption.GetSplineValue(evaluationPoints,
                                   optionTimeCoeffs, 0);
}

Console.Out.WriteLine(" Strike={0,5:f2}, Sigma={1,5:f2}, " + "Alpha={2,5:f2}",
                      strikePrices[i3 - 1],
                      sigma[i2 - 1],
                      alpha[i1 - 1]);

for (int i = 0; i < nv; i++)
{
    Console.Out.Write(" Call " +
                      "Option Values ");
    for (int j = 0; j < nt; j++)
    {
        Console.Out.Write("{0,7:f4} ", splineValuesOption[j][i]);
    }
    Console.Out.WriteLine();
}

internal class MyCoefficients : FeynmanKac.IPdeCoefficients
{
    const double zero = 0.0;
    const double half = 0.5;
    private double sigma, strikePrice, interestRate;
    private double alpha, dividend;

    public MyCoefficients(double[] myData)
    {
        this.strikePrice = myData[0];
        this.sigma = myData[2];
        this.alpha = myData[3];
        this.interestRate = myData[4];
        this.dividend = myData[5];
    }

    // The coefficient sigma(x)
    public double Sigma(double x, double t)
    {

return (sigma * Math.Pow(x, alpha * half));
}

// The coefficient derivative d(sigma) / dx
public double SigmaPrime(double x, double t)
{
    return (half * alpha * sigma * Math.Pow(x, alpha * half - 1.0));
}

// The coefficient mu(x)
public double Mu(double x, double t)
{
    return ((interestRate - dividend) * x);
}

// The coefficient kappa(x)
public double Kappa(double x, double t)
{
    return interestRate;
}

internal class MyBoundaries : FeynmanKac.IBoundaries
{
    private double xMax, df, interestRate, strikePrice;

    public MyBoundaries(double[] myData)
    {
        this.strikePrice = myData[0];
        this.xMax = myData[1];
        this.interestRate = myData[4];
    }

    public void LeftBoundaries(double time, double[,] bndCoeffs)
    {
        bndCoeffs[0, 0] = 1.0;
        bndCoeffs[0, 1] = 0.0;
        bndCoeffs[0, 2] = 0.0;
        bndCoeffs[0, 3] = 0.0;
        bndCoeffs[1, 0] = 0.0;
        bndCoeffs[1, 1] = 1.0;
        bndCoeffs[1, 2] = 0.0;
        bndCoeffs[1, 3] = 0.0;
        bndCoeffs[2, 0] = 0.0;
        bndCoeffs[2, 1] = 0.0;
        bndCoeffs[2, 2] = 1.0;
        bndCoeffs[2, 3] = 0.0;

        return;
    }

    public void RightBoundaries(double time, double[,] bndCoeffs)
    {
        df = Math.Exp(interestRate * time);
        bndCoeffs[0, 0] = 1.0;
    }
}
bndCoeffs[0, 1] = 0.0;
bndCoeffs[0, 2] = 0.0;
bndCoeffs[0, 3] = xMax - df * strikePrice;
bndCoeffs[1, 0] = 0.0;
bndCoeffs[1, 1] = 1.0;
bndCoeffs[1, 2] = 0.0;
bndCoeffs[1, 3] = 1.0;
bndCoeffs[2, 0] = 0.0;
bndCoeffs[2, 1] = 0.0;
bndCoeffs[2, 2] = 1.0;
bndCoeffs[2, 3] = 0.0;
return;
}
public double Terminal(double x)
{
    double zero = 0.0;
    // The payoff function
    double val = Math.Max(x - strikePrice, zero);
    return val;
}

Output

Constant Elasticity of Variance Model for Vanilla Call
Interest Rate: 0.050  Continuous Dividend: 0.000
Minimum and Maximum Prices of Underlying: 0.00
Number of equally spaced spline knots: 120
Number of unknowns: 363
Time in Years Prior to Expiration: 0.0833 0.3333 0.5833
Option valued at Underlying Prices: 19.00 20.00 21.00

Strike=15.00, Sigma= 0.20, Alpha= 2.00
Call Option Values  4.0624  4.2576  4.4734
Call Option Values  5.0624  5.2505  5.4492
Call Option Values  6.0624  6.2486  6.4386

Strike=20.00, Sigma= 0.20, Alpha= 2.00
Call Option Values  0.1312  0.5951  0.9693
Call Option Values  0.5024  1.0880  1.5093
Call Option Values  1.1980  1.7478  2.1745

Strike=25.00, Sigma= 0.20, Alpha= 2.00
Call Option Values  0.0000  0.0111  0.0751
Call Option Values  0.0000  0.0376  0.1630
Call Option Values  0.0006  0.1036  0.3150

Strike=15.00, Sigma= 0.30, Alpha= 2.00
Call Option Values  4.0636  4.3405  4.6627
Call Option Values  5.0625  5.2951  5.5794
Call Option Values  6.0624  6.2712  6.5248
<table>
<thead>
<tr>
<th>Strike</th>
<th>Sigma</th>
<th>Alpha</th>
<th>Call Option Values</th>
<th>Call Option Values</th>
<th>Call Option Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.00</td>
<td>0.30</td>
<td>2.00</td>
<td>0.3107 1.0261 1.5479</td>
<td>0.7317 1.5404 2.0999</td>
<td>1.3762 2.1674 2.7362</td>
</tr>
<tr>
<td>25.00</td>
<td>0.30</td>
<td>2.00</td>
<td>0.0005 0.1124 0.3564</td>
<td>0.0035 0.2184 0.5565</td>
<td>0.0184 0.3869 0.8230</td>
</tr>
<tr>
<td>15.00</td>
<td>0.40</td>
<td>2.00</td>
<td>4.0755 4.5143 4.9673</td>
<td>5.0660 5.4210 5.8328</td>
<td>6.0633 6.3588 6.7306</td>
</tr>
<tr>
<td>20.00</td>
<td>0.40</td>
<td>2.00</td>
<td>0.5109 1.4625 2.1260</td>
<td>0.9611 1.9934 2.6915</td>
<td>1.5807 2.6088 3.3202</td>
</tr>
<tr>
<td>25.00</td>
<td>0.40</td>
<td>2.00</td>
<td>0.0081 0.3302 0.7795</td>
<td>0.0287 0.5178 1.0656</td>
<td>0.0820 0.7690 1.4097</td>
</tr>
<tr>
<td>15.00</td>
<td>0.20</td>
<td>1.00</td>
<td>4.0624 4.2479 4.4312</td>
<td>5.0624 5.2479 5.4312</td>
<td>6.0624 6.2479 6.4312</td>
</tr>
<tr>
<td>20.00</td>
<td>0.20</td>
<td>1.00</td>
<td>0.0000 0.0219 0.1051</td>
<td>0.1497 0.4107 0.6484</td>
<td>1.0832 1.3314 1.5773</td>
</tr>
<tr>
<td>25.00</td>
<td>0.20</td>
<td>1.00</td>
<td>0.0000 0.0000 0.0000</td>
<td>0.0000 0.0000 0.0000</td>
<td>0.0000 0.0000 0.0000</td>
</tr>
<tr>
<td>15.00</td>
<td>0.30</td>
<td>1.00</td>
<td>4.0624 4.2479 4.4312</td>
<td>5.0624 5.2479 5.4312</td>
<td>6.0624 6.2479 6.4312</td>
</tr>
<tr>
<td>20.00</td>
<td>0.30</td>
<td>1.00</td>
<td>0.0010 0.0786 0.2208</td>
<td>0.1993 0.4997 0.7539</td>
<td>1.0835 1.3444 1.6022</td>
</tr>
<tr>
<td>25.00</td>
<td>0.30</td>
<td>1.00</td>
<td>0.0000 0.0000 0.0000</td>
<td>0.0000 0.0000 0.0000</td>
<td>0.0000 0.0000 0.0004</td>
</tr>
<tr>
<td>Strike</td>
<td>Sigma</td>
<td>Alpha</td>
<td>Call Option Values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>--------</td>
<td>----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.00</td>
<td>0.40</td>
<td>1.00</td>
<td>4.0624 4.2479 4.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.0624 5.2479 5.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6.0624 6.2479 6.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.00</td>
<td>0.40</td>
<td>1.00</td>
<td>0.0072 0.1540 0.3446</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.2498 0.5950 0.8728</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.0868 1.3795 1.6586</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.00</td>
<td>0.40</td>
<td>1.00</td>
<td>0.0000 0.0000 0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000 0.0000 0.0005</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000 0.0002 0.0057</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.00</td>
<td>0.20</td>
<td>0.00</td>
<td>4.0624 4.2479 4.4311</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.0624 5.2479 5.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6.0624 6.2479 6.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.00</td>
<td>0.20</td>
<td>0.00</td>
<td>0.0001 0.0001 0.0002</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0816 0.3316 0.5748</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.0817 1.3308 1.5748</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.00</td>
<td>0.20</td>
<td>0.00</td>
<td>0.0000 0.0000 0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000 0.0000 0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000 0.0000 0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.00</td>
<td>0.30</td>
<td>0.00</td>
<td>4.0624 4.2479 4.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.0624 5.2479 5.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6.0624 6.2479 6.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.00</td>
<td>0.30</td>
<td>0.00</td>
<td>0.0000 0.0000 0.0026</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0894 0.3326 0.5753</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.0826 1.3306 1.5749</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.00</td>
<td>0.30</td>
<td>0.00</td>
<td>0.0000 0.0000 0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000 0.0000 0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000 0.0000 0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.00</td>
<td>0.40</td>
<td>0.00</td>
<td>4.0624 4.2479 4.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.0624 5.2479 5.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6.0624 6.2479 6.4312</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.00</td>
<td>0.40</td>
<td>0.00</td>
<td>0.0000 0.0001 0.0108</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0985 0.3383 0.5781</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.0830 1.3306 1.5749</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.00</td>
<td>0.40</td>
<td>0.00</td>
<td>0.0000 0.0000 0.0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

264 • FeynmanKac  IMSL C# Numerical Library
Example 3: European Option with two payoff strategies

This example evaluates the price of a European Option using two payoff strategies: Cash-or-Nothing and Vertical Spread. In the first case the payoff function is

\[ p(x) = \begin{cases} 
0, & x \leq K \\
B, & x > K 
\end{cases} \]

The value \( B \) is regarded as the bet on the asset price, see Wilmott et al. (1995, p. 39-40). The second case has the payoff function

\[ p(x) = \max(x - K_1) - \max(x - K_2), K_2 > K_1. \]

Both problems use the same boundary conditions. Each case requires a separate integration of the Black-Scholes differential equation, but only the payoff function evaluation differs in each case. The sets of parameters in the computation are:

1. Strike and bet prices \( K_1 = 10.0, K_2 = 15.0, \) and \( B = 2.0 \).
2. Volatility \( \sigma = 0.4 \)
3. Times until expiration = \( \{1/4, 1/2\} \)
4. Interest rate \( r = 0.1 \)
5. \( x_{\text{min}} = 0.0, x_{\text{max}} = 30.0 \)
6. \( nxGrid = 61, n = 3 \times nxGrid = 183 \)

```csharp
using System;
using Imsl.Math;

class FeynmanKacEx3
{
    public static void Main(String[] args)
    {
        int i, j;
        int nxGrid = 61;
        int ntGrid = 2;
        int nv = 12;
        // The strike price
        double strikePrice = 10.0;
        // The spread value
        double spreadValue = 15.0;
        // The Bet for the Cash-or-Nothing Call
        double bet = 2.0;
        // The sigma value
```
double sigma = 0.4;
// Time values for the options
double[] tGrid = { 0.25, 0.5);
// Values of the underlying where evaluations are made
double[] evaluationPoints = new double[nv];
// Value of the interest rate and continuous dividend
double r = 0.1, dividend = 0.0;
// Values of the min and max underlying values modeled
double xMin = 0.0, xMax = 30.0;
// Define parameters for the integration step.
int nint = nxGrid - 1, n = 3 * nxGrid;
double[] xGrid = new double[nxGrid];
double dx;
// Number of left/right boundary conditions.
int nlbcd = 3, nrbcd = 3;
// Structure for the evaluation routines.
int iData;
double[] rData = new double[7];
bool[] timeDependence = new bool[7];
// Define an equally-spaced grid of points for the
// underlying price
dx = (xMax - xMin) / ((double)(nint));
xGrid[0] = xMin;
xGrid[nxGrid - 1] = xMax;
for (i = 2; i <= nxGrid - 1; i++)
{
    xGrid[i - 1] = xGrid[i - 2] + dx;
}
for (i = 1; i <= nv; i++)
{
    evaluationPoints[i - 1] = 2.0 + (i - 1) * 2.0;
}
for (i = 0; i < 7; i++)
{
    rData[i] = strikePrice;
}
rData[1] = bet;
rData[2] = spreadValue;
rData[3] = xMax;
rData[4] = sigma;
rData[5] = r;
rData[6] = dividend;
// Flag the difference in payoff functions
// 1 for the Bet, 2 for the Vertical Spread
// In both cases, no time dependencies for
// the coefficients and the left boundary
// conditions
timeDependence[5] = true;
iData = 1;
FeynmanKac betOption = new FeynmanKac(new MyCoefficients(rData));
betOption.SetTimeDependence(timeDependence);
betOption.ComputeCoefficients(nlbcd, nrbcd,
    new MyBoundaries(rData, iData), xGrid, tGrid);
FeynmanKac spreadOption = new FeynmanKac(new MyCoefficients(rData));

spreadOption.SetTimeDependence(timeDependence);
spreadOption.ComputeCoefficients(albcd, nrbcd, 
new MyBoundaries(rData, iData), xGrid, tGrid);
double[,] spreadOptionCoefficients = 
spreadOption.GetSplineCoefficients();
double[][] splineValuesSpreadOption = new double[ntGrid][];
double[] spreadOptionTimeCoeffs = new double[n];

// Evaluate solutions at vector evaluationPoints, at each time value
// prior to expiration.
for (i = 0; i < ntGrid; i++)
{
    for (j = 0; j < n; j++)
    {
        betOptionTimeCoeffs[j] = betOptionCoefficients[i + 1, j];
        spreadOptionTimeCoeffs[j] = spreadOptionCoefficients[i + 1, j];
    }

    splineValuesBetOption[i] = betOption.GetSplineValue(
evaluationPoints, betOptionTimeCoeffs, 0);
    splineValuesSpreadOption[i] = spreadOption.GetSplineValue(
evaluationPoints, spreadOptionTimeCoeffs, 0);
}

Console.Out.WriteLine(" European Option Value for A Bet");
Console.Out.WriteLine(" and a Vertical Spread, 3 and 6 Months prior to Expiry");
for (i = 0; i < nv; i++)
{
evaluationPoints[i], splineValuesBetOption[0][i], 
splineValuesBetOption[1][i], splineValuesSpreadOption[0][i], 
splineValuesSpreadOption[1][i]);
}

class MyCoefficients : FeynmanKac.IPdeCoefficients
{
    const double zero = 0.0;
    double sigma, strikePrice, interestRate;
    double spread, bet, dividend;

    // These routines define the coefficients, payoff, boundary conditions
    // and forcing term for American and European Options.
}
public MyCoefficients(double[] rData)
{
    this.strikePrice = rData[0];
    this.bet = rData[1];
    this.spread = rData[2];
    this.sigma = rData[4];
    this.interestRate = rData[5];
    this.dividend = rData[6];
}

// The coefficient sigma(x)
public double Sigma(double x, double t)
{
    return (sigma * x);
}

// The coefficient derivative d(sigma) / dx
public double SigmaPrime(double x, double t)
{
    return sigma;
}

// The coefficient mu(x)
public double Mu(double x, double t)
{
    return ((interestRate - dividend) * x);
}

// The coefficient kappa(x)
public double Kappa(double x, double t)
{
    return interestRate;
}
}

class MyBoundaries : FeynmanKac.IBoundaries
{

double strikePrice, spread, bet, interestRate, df;
int dataInt;

public MyBoundaries(double[] rData, int iData)
{
    this.strikePrice = rData[0];
    this.bet = rData[1];
    this.spread = rData[2];
    this.interestRate = rData[5];
    this.dataInt = iData;
}

public void LeftBoundaries(double time, double[,] bndCoeffs)
{
    bndCoeffs[0, 0] = 1.0;
    bndCoeffs[0, 1] = 0.0;
    bndCoeffs[0, 2] = 0.0;
}
bndCoeffs[0, 3] = 0.0;
bndCoeffs[1, 0] = 0.0;
bndCoeffs[1, 1] = 1.0;
bndCoeffs[1, 2] = 0.0;
bndCoeffs[1, 3] = 0.0;
bndCoeffs[2, 0] = 0.0;
bndCoeffs[2, 1] = 0.0;
bndCoeffs[2, 2] = 1.0;
bndCoeffs[2, 3] = 0.0;

return;
}

public void RightBoundaries(double time, double[,] bndCoeffs)
{
    // This is the discount factor using the risk-free
    // interest rate
    df = Math.Exp(interestRate * time);
    // Use flag passed to decide on boundary condition
    switch (dataInt)
    {
        case 1:
            bndCoeffs[0, 0] = 1.0;
bndCoeffs[0, 1] = 0.0;
bndCoeffs[0, 2] = 0.0;
bndCoeffs[0, 3] = bet * df;
            break;

        case 2:
            bndCoeffs[0, 0] = 1.0;
bndCoeffs[0, 1] = 0.0;
bndCoeffs[0, 2] = 0.0;
bndCoeffs[0, 3] = (spread - strikePrice) * df;
            break;
    }

    bndCoeffs[1, 0] = 0.0;
bndCoeffs[1, 1] = 1.0;
bndCoeffs[1, 2] = 0.0;
bndCoeffs[1, 3] = 0.0;
bndCoeffs[2, 0] = 0.0;
bndCoeffs[2, 1] = 0.0;
bndCoeffs[2, 2] = 1.0;
bndCoeffs[2, 3] = 0.0;

return;
}

public double Terminal(double x)
{
    const double zero = 0.0;
double val = 0.0;
    switch (dataInt)
    {
        // The payoff function - Use flag passed to decide which

case 1:
    // After reaching the strike price the payoff jumps
    // from zero to the bet value.
    val = zero;
    if (x > strikePrice)
    {
        val = bet;
    }
    break;

case 2:
    /* Function is zero up to strike price.
    Then linear between strike price and spread.
    Then has constant value Spread-Strike Price after
    the value Spread. */
    val = Math.Max(x - strikePrice, zero) -
        Math.Max(x - spread, zero);
    break;
}
return val;\n
Output

European Option Value for A Bet
and a Vertical Spread, 3 and 6 Months prior to Expiry
Number of equally spaced spline knots: 61
Number of unknowns: 183
Strike=10.00, Sigma= 0.40, Interest Rate= 0.10
Bet= 2.00, Spread Value=15.00

<table>
<thead>
<tr>
<th>Underlying</th>
<th>A Bet</th>
<th>Vertical Spread</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>4.0000</td>
<td>0.0000</td>
<td>0.0013</td>
</tr>
<tr>
<td>6.0000</td>
<td>0.0112</td>
<td>0.0729</td>
</tr>
<tr>
<td>8.0000</td>
<td>0.2686</td>
<td>0.4291</td>
</tr>
<tr>
<td>10.0000</td>
<td>0.9948</td>
<td>0.9781</td>
</tr>
<tr>
<td>12.0000</td>
<td>1.6103</td>
<td>1.4301</td>
</tr>
<tr>
<td>14.0000</td>
<td>1.8650</td>
<td>1.6926</td>
</tr>
<tr>
<td>16.0000</td>
<td>1.9335</td>
<td>1.8171</td>
</tr>
<tr>
<td>18.0000</td>
<td>1.9477</td>
<td>1.8696</td>
</tr>
<tr>
<td>20.0000</td>
<td>1.9501</td>
<td>1.8902</td>
</tr>
<tr>
<td>22.0000</td>
<td>1.9505</td>
<td>1.8979</td>
</tr>
<tr>
<td>24.0000</td>
<td>1.9506</td>
<td>1.9008</td>
</tr>
</tbody>
</table>

Example 4: Convertible bonds

This example evaluates the price of a convertible bond. Here, convertibility means that the bond may, at
any time of the holder’s choosing, be converted to a multiple of the specified asset. Thus a convertible
bond with price \( x \) returns an amount \( K \) at time \( T \) unless the owner has converted the bond to \( \nu x, \nu \geq 1 \)
units of the asset at some time prior to \( T \). This definition, the differential equation and boundary
conditions are given in Chapter 18 of Wilmott et al. (1996). Using a constant interest rate and volatility factor, the parameters and boundary conditions are:

1. Bond face value \( K = 1 \), conversion factor \( \nu = 1.125 \)
2. Volatility \( \sigma = 0.25 \)
3. Times until expiration = \{1/2, 1\}
4. Interest rate \( r = 0.1 \), dividend \( D = 0.02 \)
5. \( x_{\text{min}} = 0, x_{\text{max}} = 4 \)
6. \( nxGrid = 61, n = 3 \times nxGrid = 183 \)
7. Boundary conditions \( f(0,t) = K \exp(r - (T - t)), f(x_{\text{max}},t) = \nu x_{\text{max}} \)
8. Terminal data \( f(x,T) = \max(K, \nu x) \)
9. Constraint for bond holder \( f(x,t) \geq \nu x \)

Note that the error tolerance is set to a pure absolute error of value \( 10^{-3} \). The free boundary constraint \( f(x,t) \geq \nu x \) is achieved by use of a non-linear forcing term in interface \( \text{IForcingTerm} \). The coefficient values of the Hermite quintic spline representing the approximate solution of the differential equation at the initial time point are provided with the interface \( \text{IInitialData} \).

```csharp
using System;
using Imsl.Math;

public class FeynmanKacEx4
{
    public static void Main(String[] args)
    {
        int i, j;
        int nxGrid = 61;
        int ntGrid = 2;
        int nv = 13;

        // Compute value of a Convertible Bond
        // The face value
        double KS = 1.0;
        // The sigma or volatility value
        double sigma = 0.25;
        // Time values for the options
        double[] tGrid = { 0.5, 1.0 };  // Values of the underlying where evaluation are made
        double[] evaluationPoints = new double[nv];
        // Value of the interest rate, continuous dividend and factor
        double r = 0.1, dividend = 0.02, factor = 1.125;
        // Values of the min and max underlying values modeled
        double xmin = 0.0, xmax = 4.0;
        // Define parameters for the integration step.
        int nint = nxGrid - 1, n = 3 * nxGrid;
        double[] xGrid = new double[nxGrid];
```
// Array for user-defined data
double[] rData = new double[8];
double dx, atol;
// Number of left/right boundary conditions.
int nlbcd = 3, nrbcd = 3;
bool[] timeDependence = new bool[7];

/*
 * Define an equally-spaced grid of points for the
 * underlying price
 */
dx = (xMax - xMin) / ((double)nint);
xGrid[0] = xMin;
xGrid[nxGrid - 1] = xMax;
for (i = 2; i <= nxGrid - 1; i++)
{
    xGrid[i - 1] = xGrid[i - 2] + dx;
}
for (i = 1; i <= nv; i++)
{
    evaluationPoints[i - 1] = (i - 1) * 0.25;
}
// Pass the data for evaluation
rData[0] = KS;
rData[1] = xMax;
rData[2] = sigma;
rData[3] = r;
rData[4] = dividend;
rData[5] = factor;
// Use a pure absolute error tolerance for the integration
atol = 1.0e-3;
rData[6] = atol;
// Compute value of convertible bond
FeynmanKac convertibleBond = new FeynmanKac(new MyCoefficients(rData));

MyForcingTerm forceTerm = new MyForcingTerm(rData);
MyInitialData initialData = new MyInitialData(rData);

convertibleBond.SetForcingTerm(forceTerm);
convertibleBond.SetInitialData(initialData);
convertibleBond.SetAbsoluteErrorTolerances(1.0e-3);
convertibleBond.SetRelativeErrorTolerances(0.0);

// Only the left boundary conditions are time dependent
timeDependence[4] = true;
convertibleBond.SetTimeDependence(timeDependence);

convertibleBond.ComputeCoefficients(nlbcd, nrbcd,
    new MyBoundaries(rData), xGrid, tGrid);
double[,] bondCoefficients = convertibleBond.GetSplineCoefficients();
double[] bondTimeCoeffs = new double[n];

double[][] bondSplineValues = new double[ntGrid + 1][];

/*
 * Evaluate and display solutions at vector of points XS(:,)
 */
for (i = 0; i <= ntGrid; i++)
{
    for (j = 0; j < n; j++)
    
        bondTimeCoeffs[j] = bondCoefficients[i, j];

        bondSplineValues[i] = convertibleBond.GetSplineValue(
        evaluationPoints, bondTimeCoeffs, 0);
}

Console.Out.WriteLine(" Convertible Bond Value, 0+, 6 and 12 Months "
 + "Prior to Expiry", " ");
Console.Out.WriteLine(" Number of equally spaced spline knots:{0,4:d}", nxGrid);
Console.Out.WriteLine(" Number of unknowns:{0,4:d}", n);
Console.Out.WriteLine(" Strike={0,5:f2}, Sigma={1,5:f2}", 
KS, sigma);
Console.Out.WriteLine(" Interest Rate={0,5:f2}, " + 
"Dividend={1,5:f2}, Factor={0,6:f3}" , r, dividend, factor);
Console.Out.WriteLine(" Underlying Bond Value");
for (i = 0; i < nv; i++)
{
    Console.Out.WriteLine(" {0,8:f4}{1,8:f4}{2,8:f4}{3,8:f4}",
    evaluationPoints[i],
    bondSplineValues[0][i],
    bondSplineValues[1][i],
    bondSplineValues[2][i]);
}

class MyCoefficients : FeynmanKac.IPdeCoefficients
{
    double sigma, strikePrice, interestRate;
    double dividend, factor;

    public MyCoefficients(double[] rData)
    {
        this.strikePrice = rData[0];
        this.sigma = rData[2];
        this.interestRate = rData[3];
        this.dividend = rData[4];
        this.factor = rData[5];
    }

    // The coefficient sigma(x)
    public double Sigma(double x, double t)
    {
        return (sigma * x);
    }
}
public double SigmaPrime(double x, double t) {
    return sigma;
}

public double Mu(double x, double t) {
    return ((interestRate - dividend) * x);
}

public double Kappa(double x, double t) {
    return interestRate;
}

class MyBoundaries : FeynmanKac.IBoundaries {

    double interestRate, strikePrice, dp, factor, xMax;

    public MyBoundaries(double[] rData) {
        this.strikePrice = rData[0];
        this.xMax = rData[1];
        this.interestRate = rData[3];
        this.factor = rData[5];
    }

    public void LeftBoundaries(double time, double[,] bndCoeffs) {
        dp = strikePrice * Math.Exp(time * interestRate);
        bndCoeffs[0, 0] = 1.0;
        bndCoeffs[0, 1] = 0.0;
        bndCoeffs[0, 2] = 0.0;
        bndCoeffs[0, 3] = dp;
        bndCoeffs[1, 0] = 0.0;
        bndCoeffs[1, 1] = 1.0;
        bndCoeffs[1, 2] = 0.0;
        bndCoeffs[1, 3] = 0.0;
        bndCoeffs[2, 0] = 0.0;
        bndCoeffs[2, 1] = 0.0;
        bndCoeffs[2, 2] = 1.0;
        bndCoeffs[2, 3] = 0.0;

        return;
    }

    public void RightBoundaries(double time, double[,] bndCoeffs) {
        bndCoeffs[0, 0] = 1.0;
        bndCoeffs[0, 1] = 0.0;
        bndCoeffs[0, 2] = 0.0;
    }
}
bndCoeffs[0, 3] = factor * xMax;
bndCoeffs[1, 0] = 0.0;
bndCoeffs[1, 1] = 1.0;
bndCoeffs[1, 2] = 0.0;
bndCoeffs[1, 3] = factor;
bndCoeffs[2, 0] = 0.0;
bndCoeffs[2, 1] = 0.0;
bndCoeffs[2, 2] = 1.0;
bndCoeffs[2, 3] = 0.0;

    return;
}

public double Terminal(double x)
{
    // The payoff function
    double val = Math.Max(factor * x, strikePrice);
    return val;
}

class MyForcingTerm : FeynmanKac.IForcingTerm
{
    const double zero = 0.0;
    const double one = 1.0;
    double val, strikePrice, interestRate;
    double rt, mu, factor;

    public MyForcingTerm(double[] rData)
    {
        this.val = rData[6];
        this.strikePrice = rData[0];
        this.interestRate = rData[3];
        this.factor = rData[5];
    }

    public void Force(int interval, double[] y, double time, double width,
                       double[] xlocal, double[] qw, double[,] u,
                       double[] phi, double[,] dphi)
    {
        const int local = 6;
        int ndeg = xlocal.Length;
        double[] yl = new double[6];
        double[] bf = new double[6];
        for (int i = 0; i < local; i++)
        {
            yl[i] = y[3 * interval - 3 + i];
            phi[i] = zero;
        }
        for (int i = 0; i < local; i++)
        {
            for (int j = 0; j < local; j++)
            {
                dphi[i, j] = zero;
            }
        }
    }
}
mu = 2.0;

/*
 * This is the local definition of the forcing term -
 * It "forces" the constraint f >= factor*x.
 */
for (int j = 1; j <= local; j++)
{
    for (int l = 1; l <= ndeg; l++)
    {
        bf[0] = u[0, l - 1];
        bf[1] = u[1, l - 1];
        bf[2] = u[2, l - 1];
        bf[3] = u[3, l - 1];
        bf[4] = u[4, l - 1];
        bf[5] = u[5, l - 1];
        rt = 0.0;
        for (int k = 0; k < local; k++)
        {
            rt += yl[k] * bf[k];
        }
        rt = val / (rt + val - factor * xlocal[l - 1]);
    }
}
for (int i = 0; i < local; i++)
{
    phi[i] = (-phi[i]) * width * factor * strikePrice;
}

/*
 * This is the local derivative matrix for the forcing term
 */
for (int j = 1; j <= local; j++)
{
    for (int i = 1; i <= local; i++)
    {
        for (int l = 1; l <= ndeg; l++)
        {
            bf[0] = u[0, l - 1];
            bf[1] = u[1, l - 1];
            bf[2] = u[2, l - 1];
            bf[3] = u[3, l - 1];
            bf[4] = u[4, l - 1];
            bf[5] = u[5, l - 1];
            rt = 0.0;
            for (int k = 0; k < local; k++)
            {
                rt += yl[k] * bf[k];
            }
            rt = one / (rt + val - factor * xlocal[l - 1]);
            dphi[j - 1, i - 1] += qw[l - 1] * bf[i - 1] * bf[j - 1] *
                                Math.Pow(val * rt, mu) * rt;
        }
    }
}
for (int i = 0; i < local; i++)
{  
    for (int j = 0; j < local; j++)  
    {  
        dphi[i, j] = (-mu) * dphi[i, j] * width * factor *  
                     strikePrice;  
    }  
    return;  
  }  
}

class MyInitialData : FeynmanKac.IInitialData  
{
    private double[] data;
    
    public MyInitialData(double[] rData)  
    {  
        data = new double[rData.Length];  
        for (int i = 0; i < rData.Length; i++)  
        {  
            data[i] = rData[i];  
        }  
    }  
    
    public void Init(double[] xGrid, double[] tGrid, double tp,  
                      double[] yprime, double[] y, double[] atol, double[] rtol)  
    {  
        int nxGrid = xGrid.Length;  
        if (tp == 0.0)  
            // Set initial data precisely.  
            {  
                for (int i = 1; i <= nxGrid; i++)  
                {  
                    if (xGrid[i - 1] * data[5] < data[0])  
                    {  
                        y[3 * i - 3] = data[0];  
                        y[3 * i - 2] = 0.0;  
                        y[3 * i - 1] = 0.0;  
                    }  
                    else  
                    {  
                        y[3 * i - 3] = xGrid[i - 1] * data[5];  
                        y[3 * i - 2] = data[5];  
                        y[3 * i - 1] = 0.0;  
                    }  
                }  
                return;  
            }  
    }  
}
Example 5: Calculating the Greeks using the Feynman-Kac Class

In this example, the Feynman-Kac (FK) class is used to solve for the Greeks, i.e. various derivatives of FK solutions applicable to the pricing of options and related financial derivatives. In order to illustrate and verify these calculations, the Greeks are calculated by two methods. The first method involves the FK solution to the diffusion model for call options given in Example 2 for the Black-Scholes (BS) case, i.e. \( \alpha = 2 \). The second method calculates the Greeks using the closed-form BS evaluations which can be found at http://en.wikipedia.org/wiki/The_Greeks.

This example calculates FK and BS solutions \( V(S, t) \) to the BS problem and the following Greeks:

- **Delta** = \( \frac{\partial V}{\partial S} \) is the first derivative of the Value, \( V(S, t) \), of a portfolio of derivative security derived from underlying instrument with respect to the underlying instrument’s price \( S \);
- **Gamma** = \( \frac{\partial^2 V}{\partial S^2} \);
- **Theta** = \( -\frac{\partial V}{\partial t} \) is the negative first derivative of \( V \) with respect to time \( t \);
- **Charm** = \( \frac{\partial^2 V}{\partial S \partial t} \);
- **Color** = \( \frac{\partial^3 V}{\partial S^2 \partial t} \);
- **Rho** = \( -\frac{\partial V}{\partial r} \) is the first derivative of \( V \) with respect to risk free rate \( r \);
- **Vega** = \( \frac{\partial V}{\partial \sigma} \) measures sensitivity to volatility parameter \( \sigma \) of the underlying \( S \);
- **Volga** = \( \frac{\partial^2 V}{\partial \sigma^2} \);
- **Vanna** = \( \frac{\partial^3 V}{\partial S \partial \sigma^2} \).
Intrinsic Greeks include derivatives involving only \( S \) and \( t \), the intrinsic FK arguments. In the above list, Value, Delta, Gamma, Theta, Charm, Color, and Speed are all intrinsic Greeks. As is discussed in Hanson, R. (2008) *Integrating Feynman-Kac equations Using Hermite Quintic Finite Elements*, the expansion of the FK solution function \( V(S, t) \) in terms of quintic polynomial functions defined on \( S \)-grid subintervals and subject to continuity constraints in derivatives 0, 1, and 2 across the boundaries of these subintervals allows Value, Delta, Gamma, Theta, Charm, and Color to be calculated directly by the FK class methods GetSplineCoefficients, GetSplineCoefficientsPrime, and GetSplineValue.

Non-intrinsic Greeks are derivatives of \( V \) involving FK parameters other than the intrinsic arguments \( S \) and \( t \), such as \( r \) and \( \sigma \). Non-intrinsic Greeks in the above list include Rho, Vega, Volga, and Vanna. In order to calculate non-intrinsic Greek (parameter) derivatives or intrinsic Greek \( S \)-derivatives beyond the second (such as Speed) or \( t \)-derivatives beyond the first, the entire FK solution must be calculated 3 times (for a parabolic fit) or five times (for a quartic fit), at the point where the derivative is to be evaluated and at nearby points located symmetrically on either side.

Using a Taylor series expansion of \( f(\sigma + \epsilon) \) truncated to \( m+1 \) terms (to allow an \( m \)-degree polynomial fit of \( m+1 \) data points):

\[
f(\sigma + \epsilon) = \sum_{n=0}^{m} \frac{f^{(n)}(\sigma)}{n!} \epsilon^n
\]

we are able to derive the following parabolic (3 point) estimation of first and second derivatives \( f^{(1)}(\sigma) \) and \( f^{(2)}(\sigma) \) in terms of the three values: \( f(\sigma - \epsilon), f(\sigma), \) and \( f(\sigma + \epsilon) \), where \( \epsilon = \epsilon_{frac} \cdot \sigma \) and \( 0 < \epsilon_{frac} < 1 \):

\[
f^{(1)}(\sigma) \equiv \frac{\partial f(\sigma)}{\partial \sigma} \approx f^{[1]}(\sigma, \epsilon) \equiv \frac{f(\sigma + \epsilon) - f(\sigma - \epsilon)}{2 \epsilon}
\]

\[
f^{(2)}(\sigma) \equiv \frac{\partial^2 f(\sigma)}{\partial \sigma^2} \approx f^{[2]}(\sigma, \epsilon) \equiv \frac{f(\sigma + \epsilon) + f(\sigma - \epsilon) - 2 f(\sigma)}{\epsilon^2}
\]

Similarly, the quartic (5 point) estimation of \( f^{(1)}(\sigma) \) and \( f^{(2)}(\sigma) \) in terms of \( f(\sigma - 2\epsilon), f(\sigma - \epsilon), f(\sigma), f(\sigma + \epsilon), \) and \( f(\sigma + 2\epsilon) \) is:

\[
f^{(1)}(\sigma) \approx \frac{4}{3} f^{[1]}(\sigma, \epsilon) - \frac{1}{3} f^{[1]}(\sigma, 2\epsilon)
\]

\[
f^{(2)}(\sigma) \approx \frac{4}{3} f^{[2]}(\sigma, \epsilon) - \frac{1}{3} f^{[2]}(\sigma, 2\epsilon)
\]

For our example, the quartic estimate does not appear to be significantly better than the parabolic estimate, so we have produced only parabolic estimates by setting variable \( i_{quart} \) to 0. The user may try the example with the quartic estimate simply by setting \( i_{quart} \) to 1.
As is pointed out in *Integrating Feynman-Kac equations Using Hermite Quintic Finite Elements*, the quintic polynomial expansion function used by Feynman-Kac only allows for continuous derivatives through the second derivative. While up to fifth derivatives can be calculated from the quintic expansion (indeed class FeynmanKac method GetSplineValue will allow the third derivative to be calculated by setting parameter ideriv to 3, as is done in this example), the accuracy is compromised by the inherent lack of continuity across grid points (i.e. subinterval boundaries).

The accurate second derivatives in $S$ returned by FK method GetSplineValue can be leveraged into a third derivative estimate by calculating three FK second derivative solutions, the first solution for grid and evaluation point sets $\{S, f''(S)\}$ and the second and third solutions for solution grid and evaluation point sets $\{S + \epsilon, f''(S + \epsilon)\}$ and $\{S - \epsilon, f''(S - \epsilon)\}$, where the solution grid and evaluation point sets are shifted up and down by $\epsilon$. In this example, $\epsilon$ is set to $\epsilon_{frac}\bar{S}$, where $\bar{S}$ is the average value of $S$ over the range of grid values and $0 < \epsilon_{frac} << 1$. The third derivative solution can then be obtained using the parabolic estimate:

$$f^{(3)}(S) = \frac{\partial f^{(2)}(S)}{\partial S} \approx \frac{f^{(2)}(S + \epsilon) - f^{(2)}(S - \epsilon)}{2\epsilon}$$

This procedure is implemented in the current example to calculate the Greek *Speed*. (For comparison purposes, *Speed* is also calculated directly by setting the GetSplineValue input S derivative parameter iSDeriv to 3. The output from this direct calculation is called “*Speed2*”.)

The average and maximum relative errors (defined as the absolute value of the difference between the BS and FK values divided by the BS value) for each of the Greeks is given at the end of the output. (These relative error statistics are given for nine combinations of Strike Price and Volatility, but only one of the nine combinations is actually printed in the output.) Both intrinsic and non-intrinsic Greeks have good accuracy (average relative error is in the range 0.01 – 0.0001) except for Volga, which has an average relative error of about 0.05. This is probably a result of the fact that Volga involves differences of differences, which will more quickly erode accuracy than calculations using only one difference to approximate a derivative. Possible ways to improve upon the 2 to 4 significant digits of accuracy achieved in this example include increasing FK integration accuracy by reducing the initial step size (using property InitialStepsize), by choosing more closely spaced $S$ and $t$ grid points (by adjusting method ComputeCoefficients input parameter arrays xGrid and tGrid), and by adjusting $\epsilon_{frac}$ so that the central differences used to calculate the derivatives are not too small to compromise accuracy.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;

public class FeynmanKacEx5
{
    private static double[] strikePrices = { 15.0, 20.0, 25.0 }; // The set of sigma values
    private static double[] sigma = { 0.2, 0.3, 0.4 }; // The set of model diffusion powers: alpha = 2.0 <-> Black Scholes
    private static double[] alpha = { 2.0, 1.0, 0.0 }; // Time values for the options
    private static double[] tGrid = { 1.0 / 12.0, 4.0 / 12.0, 7.0 / 12.0 }; // Values of the min and max underlying values modeled
    private static double xMin = 0.0;
    ```
private static double xMax = 60.0;
// Value of the interest rate and continuous dividend
private static double r = 0.05;
private static double dividend = 0.0;
// Define parameters for the integration step.
private static int nXGgrid = 121;
private static int nTGrid = 3;
private static int nv = 3;
private static int nint = nXGgrid - 1;
private static int n = 3 * nXGgrid;
private static double[] xGrid = new double[nXGgrid];
private static double dx;
// Time dependency
private static bool[] timeDependence = new bool[7];
// Number of left/right boundary conditions
private static int nlbcd = 3;
private static int nrbcd = 3;
// Values of the underlying price where evaluations are made
private static double[] evaluationPoints = { 19.0, 20.0, 21.0 }; // used to calc derivatives
private static double epsfrac = 0.001;
private static double dx2 = epsfrac * 0.5 * (xMin + xMax);
private static double sqrt2pi = Math.Sqrt(2.0 * Math.PI);
private static String[] greekName = new String[]{
    " Value", " Delta", " Gamma", " Theta",
    " Charm", " Color", " Vega", " Volga",
    " Vanna", " Rho", " Speed", " Speed2"
};

// Time values for the options
private static double[] rex = new double[greekName.Length];
private static double[] reavg = new double[greekName.Length];
private static int[] irect = new int[greekName.Length];

// Compute Constant Elasticity of Variance Model for Vanilla Call
public static void Main(String[] args)
{
    // Define equally-spaced grid of points for the underlying price
    double dx = (xMax - xMin) / (double)nint;
    xGrid[0] = xMin;
    xGrid[nXGgrid - 1] = xMax;

    for (int i = 2; i < nXGgrid - 1; i++)
    {
        xGrid[i] = xGrid[i - 2] + dx;
    }
    Console.Out.WriteLine(" Constant Elasticity of Variance Model" +
                        " for Vanilla Call Option");
    Console.Out.WriteLine(" Interest Rate:{0,7:f3} Continuous Dividend:{1,7:f3}" ,
                        r, dividend);
    Console.Out.WriteLine(" Minimum and Maximum Prices of Underlying:{0,7:f2} - {1,7:f2}" ,
                        xMin, xMax);
    Console.Out.WriteLine(" Number of equally spaced spline knots:{0,4:d}" ,
                        nXGgrid - 1);
Console.Out.WriteLine(" Number of unknowns:{0,4:d}", n);
Console.Out.WriteLine();
Console.Out.WriteLine("Time in Years Prior to Expiration: {0,7:f4}{1,7:f4}{2,7:f4}",
    tGrid[0], tGrid[1], tGrid[2]); // tGrid[] = tau == T - t
Console.Out.WriteLine("Option valued at Underlying Prices:{0,7:f2}{1,7:f2}{2,7:f2}",
    evaluationPoints[0], evaluationPoints[1], evaluationPoints[2]);
Console.Out.WriteLine();

/*
iquart = 0 : derivatives estimated with 3-point fitted parabola
iquart = 1 : derivatives estimated with 5-point fitted quartic
*/
int iquart = 0;
if (iquart == 0)
{
    Console.Out.WriteLine("3 point (parabolic) estimate of parameter derivatives;");
}
else
{
    Console.Out.WriteLine("5 point (quartic) estimate of parameter derivatives");
}
Console.Out.WriteLine(" epsfrac = {0,11:f8}", epsfrac);
alpha: Black-Scholes
for (int i2 = 1; i2 <= 3; i2++)
    /* Loop over volatility */
    for (int i3 = 1; i3 <= 3; i3++)
        /* Loop over strike price */
        calculateGreeks(i2, i3, iquart);
for (int ig = 0; ig < 12; ig++)
{
    relavg[ig] /= irect[ig];
    Console.Out.WriteLine();
    Console.Out.Write(" Greek: " + greekName[ig] + "; avg rel err: {0,15:f12}; max rel err: {1,15:f12}",
        relavg[ig], rex[ig]);
}
Console.Out.WriteLine();
} // end main

private static void calculateGreeks(
    int volatility, int strikePrice, int iquart)
{
    int i1 = 1;
    int nt = 3;
    if ((volatility == 1) && (strikePrice == 1))
    {
        Console.Out.WriteLine();
        Console.Out.WriteLine(" Strike={0,5:f2}, Sigma=" +
            "{1,5:f2}, Alpha={2,5:f2}", strikePrices[strikePrice - 1],
            sigma, alpha);
sigma[volatility - 1], alpha[i1 - 1]);
Console.Out.WriteLine();
Console.Out.WriteLine(" years to expiration: " +
    " \{0,7:f4\} \{1,7:f4\} \{2,7:f4\}",
    tGrid[0], tGrid[1], tGrid[2]);
Console.Out.WriteLine();
/* Loop over t derivative index */
for (int iTDeriv = 0; iTDeriv < 2; iTDeriv++)
{
  int iSDerivMax = 4 - iTDeriv;
  /* Loop over S derivative index */
  for (int iSDeriv = 0; iSDeriv < iSDerivMax; iSDeriv++)
  {
    int iSigDerivMax = 1;
    if (iTDeriv == 0)
    {
      if (iSDeriv == 0)
        iSigDerivMax = 3;
      if (iSDeriv == 1)
        iSigDerivMax = 2;
    }
    //Loop over sigma deriv index
    for (int iSigDeriv = 0; iSigDeriv < iSigDerivMax; iSigDeriv++)
    {
      int iRDerivMax = 1;
      if (iTDeriv == 0 & iSDeriv == 0 & iSigDeriv == 0)
        iRDerivMax = 2;
      // Loop over r derivative index
      for (int iRDeriv = 0; iRDeriv < iRDerivMax; iRDeriv++)
      {
        int ispeedmin = 0;
        int ispeedmax = 1;
        if (iTDeriv == 0 & iSDeriv == 2)
          ispeedmax = 2;
        if (iTDeriv == 0 & iSDeriv == 3)
        {
          ispeedmin = 2;
          ispeedmax = 3;
        }
        // Loop over speed index
        for (int ispeed = ispeedmin; ispeed < ispeedmax; ispeed++)
        {
          // Pass data through into evaluation routines.
          double[] userData = new double[6];
          userData[0] = strikePrices[strikePrice - 1];
          userData[1] = xMax;
          userData[2] = sigma[volatility - 1];
          userData[3] = alpha[i1 - 1];
          userData[4] = r;
          userData[5] = dividend;
          double[][] splineValuesOption = new double[nTGrid][];
          for (int i = 0; i < nTGrid; i++)
          {
            splineValuesOption[i] = new double[nv];
          }
        }
    }
  }
}
double[,] splineValuesOptionP = new double[nTGrid][];
double[,] splineValuesOptionM = new double[nTGrid][];
double[,] splineValuesOptionPP = new double[nTGrid][];
double[,] splineValuesOptionMM = new double[nTGrid][];
double[] xGridP = new double[nXGgrid];
double[] xGridM = new double[nXGgrid];
double[] xGridPP = new double[nXGgrid];
double[] xGridMM = new double[nXGgrid];
double[] evaluationPointsP = new double[nv];
double[] evaluationPointsM = new double[nv];
double[] evaluationPointsPP = new double[nv];
double[] evaluationPointsMM = new double[nv];
double xMaxP = xMax;
double xMaxM = xMax;
double xMaxPP = xMax;
double xMaxMM = xMax;

// Evaluate FK solution at vector evaluationPoints, at // each time value prior to expiration.
if ((iSigDeriv != 1 || iSDeriv == 1 || iRDeriv != 1) &&
    (ispeed == 0 || ispeed == 2))
{
    FeynmanKac callOption =
        new FeynmanKac(new MyCoefficients(userData));
    //Right boundary condition time-dependent
    timeDependence[5] = true;
    callOption.SetTimeDependence(timeDependence);
    callOption.ComputeCoefficients(nlbcd, nrbcd,
        new MyBoundaries(userData), xGrid, tGrid);
    double[] tmpArray = new double[3 * xGrid.Length];
    double[, ] optionCoefficients;
    if (iTDeriv == 0)
    {
        optionCoefficients =
            callOption.GetSplineCoefficients();
    }
    else
    {
        optionCoefficients =
            callOption.GetSplineCoefficientsPrime();
    }
    for (int i = 0; i < nTGrid; i++)
    {
        for (int j = 0; j < 3 * xGrid.Length; j++)
            tmpArray[j] = optionCoefficients[i + 1, j];
        // FK option values for tau = tGrid[i]:
        splineValuesOption[i] =
            callOption.GetSplineValue(evaluationPoints,
                tmpArray, iSDeriv);
    }
    if (iSigDeriv > 0 || iRDeriv > 0 || ispeed == 1)
    {
        Array.Copy(xGrid, 0, xGridP, 0, nXGgrid);
        Array.Copy(xGrid, 0, xGridM, 0, nXGgrid);
Array.Copy(evaluationPoints, 0, evaluationPointsP, 0, nv);
Array.Copy(evaluationPoints, 0, evaluationPointsM, 0, nv);
if (ispeed == 1)
{
    for (int i = 0; i < nXGgrid; i++)
    {
        xGridP[i] = xGrid[i] + dx2;
        xGridM[i] = xGrid[i] - dx2;
    }
    for (int i = 0; i < nv; i++)
    {
        evaluationPointsP[i] = evaluationPoints[i] + dx2;
        evaluationPointsM[i] = evaluationPoints[i] - dx2;
    }
    xMaxP = xMax + dx2;
    xMaxM = xMax - dx2;
}

userData[1] = xMaxP;
// calculate spline values for
// sigmaP = sigma[i2-1]*(1. + epsfrac)
// or rP = r*(1. + epsfrac)
if (iSigDeriv > 0)
    userData[2] = sigma[volatility - 1] * (1.0 + epsfrac);  
else if (iRDeriv > 0)
    userData[4] = r * (1.0 + epsfrac);
FeynmanKac callOptionP = new FeynmanKac(new MyCoefficients(userData));
//Right boundary condition time-dependent
timeDependence[5] = true;
callOptionP.SetTimeDependence(timeDependence);
callOptionP.ComputeCoefficients(nlbcd, nrbcd,
    new MyBoundaries(userData), xGridP, tGrid);
    double[,] optionCoefficientsP;
    double[] tmpArray2 = new double[3 * xGrid.Length];
    if (iTDeriv == 0)
    {
        optionCoefficientsP =
            callOptionP.GetSplineCoefficients();
    }
    else
    {
        optionCoefficientsP =
            callOptionP.GetSplineCoefficientsPrime();
    }
    for (int i = 0; i < nTGrid; i++)
    {
        for (int j = 0; j < 3 * xGrid.Length; j++)
            tmpArray2[j] = optionCoefficientsP[i + 1, j];
            // FK option values for tau = tGrid[i]:
            splineValuesOptionP[i] =
callOptionP.GetSplineValue(evaluationPointsP, tmpArray2, iSDeriv);
}

userData[1] = xMaxM;
// calculate spline values for
// sigmaM = sigma[i2-1]*(1. - epsfrac) or
// rM = r*(1. - epsfrac):
if (iSigDeriv > 0)
  userData[2] = sigma[volatility - 1] *
                 (1.0 - epsfrac);
else
  userData[4] = r * (1.0 - epsfrac);
FeynmanKac callOptionM =
  new FeynmanKac(new MyCoefficients(userData));
// Right boundary condition time-dependent
timeDependence[5] = true;
callOptionM.SetTimeDependence(timeDependence);
callOptionM.ComputeCoefficients(nlbcd, nrbcd,
    new MyBoundaries(userData), xGridM, tGrid);
double[,] optionCoefficientsM;
double[] tmpArray3 = new double[3 * xGrid.Length];
if (iTDeriv == 0)
{
  optionCoefficientsM =
    callOptionM.GetSplineCoefficients();
}
else
{
  optionCoefficientsM =
    callOptionM.GetSplineCoefficientsPrime();
}
for (int i = 0; i < nTGrid; i++)
{
  for (int j = 0; j < 3 * xGrid.Length; j++)
    tmpArray3[j] = optionCoefficientsM[i + 1, j];
  // FK option values for tau = tGrid[i]:
  splineValuesOptionM[i] =
    callOptionM.GetSplineValue(evaluationPointsM,
                              tmpArray3, iSDeriv);
}
if (iquart == 1)
{
  Array.Copy(xGrid, 0, xGridPP, 0, nXGgrid);
  Array.Copy(xGrid, 0, xGridMM, 0, nXGgrid);
  Array.Copy(evaluationPoints, 0, evaluationPointsPP, 0, nv);
  Array.Copy(evaluationPoints, 0, evaluationPointsMM, 0, nv);
  if (ispeed == 1)
  {
    for (int i = 0; i < nXGgrid; i++)
    {
      xGridPP[i] = xGrid[i] + 2.0 * dx2;
      xGridMM[i] = xGrid[i] - 2.0 * dx2;
    }
  }
}
for (int i = 0; i < nv; i++)
{
    evaluationPointsPP[i] =
        evaluationPoints[i] + 2.0 * dx2;
    evaluationPointsMM[i] =
        evaluationPoints[i] - 2.0 * dx2;
}
xMaxPP = xMax + 2.0 * dx2;
xMaxMM = xMax - 2.0 * dx2;
}

userData[1] = xMaxPP;
if (iSigDeriv > 0)
{
    // calculate spline values for
    // sigmaPP = sigma[i2-1]*(1. + 2.*epsfrac):
    userData[2] = sigma[volatility - 1] *
        (1.0 + 2.0 * epsfrac);
}
else if (iRDeriv > 0)
{
    // calculate spline values for
    // rPP = r*(1. + 2.*epsfrac):
    userData[4] = r * (1.0 + 2.0 * epsfrac);
}

FeynmanKac callOptionPP =
    new FeynmanKac(new MyCoefficients(userData));
//Right boundary condition time-dependent
timeDependence[5] = true;
callOptionPP.SetTimeDependence(timeDependence);
callOptionPP.ComputeCoefficients(nlbcd, nrbcd,
    new MyBoundaries(userData), xGridPP, tGrid);

double[] tmpArray4 = new double[3 * xGrid.Length];
double[,] optionCoefficientsPP;
if (iTDeriv == 0)
{
    optionCoefficientsPP =
        callOptionPP.GetSplineCoefficients();
}
else
{
    optionCoefficientsPP =
        callOptionPP.GetSplineCoefficientsPrime();
}
for (int i = 0; i < nTGrid; i++)
{
    for (int j = 0; j < 3 * xGrid.Length; j++)
        tmpArray4[j] =
            optionCoefficientsPP[i + 1, j];
    // FK option values for tau = tGrid[i]:
splineValuesOptionPP[i] =
        callOptionPP.GetSplineValue(
            evaluationPointsPP, tmpArray4, iSDeriv);
 UserData[1] = xMaxMM;
// calculate spline values for
// sigmaMM = sigma[i2-1]*(1. - 2.*epsfrac) or
// rMM = r*(1. - 2.*epsfrac)
if (iSigDeriv > 0)
    UserData[2] = sigma[volatility - 1] *
                 (1.0 - 2.0 * epsfrac);
else if (iRDeriv > 0)
    UserData[4] = r * (1.0 - 2.0 * epsfrac);
FeynmanKac callOptionMM =
    new FeynmanKac(new MyCoefficients(UserData));
// Right boundary condition time-dependent
timeDependence[5] = true;
callOptionMM.SetTimeDependence(timeDependence);
callOptionMM.ComputeCoefficients(nlbcd, nrbcd,
    new MyBoundaries(UserData), xGridMM, tGrid);
double[] tmpArray5 = new double[3 * xGrid.Length];
optionCoefficientsMM =
    callOptionMM.GetSplineCoefficientsPrime();
for (int i = 0; i < nTGrid; i++)
    {  
        for (int j = 0; j < 3 * xGrid.Length; j++)
            tmpArray5[j] =
                optionCoefficientsMM[i + 1, j];
        // FK option values for tau = tGrid[i]:
        splineValuesOptionMM[i] =
            callOptionMM.GetSplineValue(      
                evaluationPointsMM, tmpArray5, iSDeriv);    
    }

if (iSigDeriv == 1 || iRDeriv == 1 || ispeed == 1)
    {
        double eps = 0.0, f11 = 0.0, f12 = 0.0;
        if (iSigDeriv == 1)
            eps = sigma[volatility - 1] * epsfrac;
        if (iRDeriv == 1)
            eps = r * epsfrac;
        if (ispeed == 1)
            eps = dx2;
        for (int i = 0; i < nTGrid; i++)
            {
                for (int j = 0; j < nv; j++)
                    {  

\begin{verbatim}
f11 = (splineValuesOptionP[i][j] - splineValuesOptionM[i][j]) / (2.0 * eps);
if (iquart == 0)
splineValuesOption[i][j] = f11;
else {
    f12 = (splineValuesOptionPP[i][j] - splineValuesOptionMM[i][j]) / (4.0 * eps);
splineValuesOption[i][j] = (4.0 * f11 - f12) / 3.0;
}
}

if (iSigDeriv == 2)
{
    double eps = sigma[volatility - 1] * epsfrac;
double f21 = 0.0;
double f22 = 0.0;
for (int i = 0; i < nTGrid; i++)
{
    for (int j = 0; j < nv; j++)
    {
        f21 = (splineValuesOptionP[i][j] +
               splineValuesOptionM[i][j] - 2.0 *
               splineValuesOption[i][j]) / (eps * eps);
        if (iquart == 0)
            splineValuesOption[i][j] = f21;
        else
        {
            f22 = (splineValuesOptionPP[i][j] +
                   splineValuesOptionMM[i][j] - 2.0 *
                   splineValuesOption[i][j]) /
                  (4.0 * eps * eps);
            splineValuesOption[i][j] =
                (4.0 * f21 - f22) / 3.0;
        }
    }
}
}

// Evaluate BS solution at vector evaluationPoints,
// at each time value prior to expiration.

double[][] BSValuesOption = new double[nTGrid][];
for (int i7 = 0; i7 < nTGrid; i7++)
{
    BSValuesOption[i7] = new double[nv];
}
for (int i = 0; i < nTGrid; i++)
{
    /*
    * Black-Scholes (BS) European call option
    * value = ValBSEC(S,t) = exp(-q*tau)*S*N01CDF(d1) -
    */
}
\end{verbatim}
\[
\exp(-r \cdot \tau) \cdot K \cdot N(0,1)CDF(d_2),
\]

where:
- \( \tau \) = time to maturity = \( T - t \);
- \( q \) = annual dividend yield;
- \( r \) = risk free rate;
- \( K \) = strike price;
- \( S \) = stock price;
- \( N(0,1)CDF(x) \) = \( N(0,1)_CDF(x) \);
- \( d_1 \) = \( \frac{\log(S/K) + (r - q + 0.5 \cdot \sigma^2) \cdot \tau}{\sigma \cdot \sqrt{\tau}} \);
- \( d_2 \) = \( d_1 - \sigma \cdot \sqrt{\tau} \).

// BS option values for \( \tau = \text{tGrid}[i] \):
double tau = tGrid[i];
double sigsqtau =
    Math.Pow(sigma[volatility - 1], 2) * tau;
double sqrt_sigsqtau = Math.Sqrt(sigsqtau);
double sigsq = sigma[volatility - 1] *
    sigma[volatility - 1];
for (int j = 0; j < nv; j++)
{
    // Values of the underlying price where
    // evaluations are made:
double S = evaluationPoints[j];
double d1 = (Math.Log(S /
        strikePrices[strikePrice - 1]) + (r - dividend)
        * tau + 0.5 * sigsqtau) / sqrt_sigsqtau;
double n01pdf_d1 =
    Math.Exp((-0.5) * d1 * d1) / sqrt2pi;
double nu = Math.Exp((-dividend) * tau) *
    S * n01pdf_d1 * Math.Sqrt(tau);
    if (iTDeriv == 0)
    {
        if (iSDeriv == 0)
        {
            double d2 = d1 - sqrt_sigsqtau;
            if (iSigDeriv == 0)
            {
                if (iRDeriv == 0)
                {
                    BSValuesOption[i][j] =
                        Math.Exp((-dividend) * tau) * S *
                        Cdf.Normal(d1) -
                        Math.Exp((-r) * tau) *
                        strikePrices[strikePrice - 1] *
                        Cdf.Normal(d2);
                }
            else
            {
                BSValuesOption[i][j] =
                    Math.Exp((-r) * tau) *
                    strikePrices[strikePrice - 1] *
                    tau * Cdf.Normal(d2);
            }
        }
    }

    // BS Values Option[i][j] =
    Math.Exp((-r) * tau) *
    strikePrices[strikePrice - 1] *
    Cdf.Normal(d2);
}

290  \textit{FeynmanKac}  \textbf{IMSL C# Numerical Library}
else if (iSigDeriv == 1)
{
    //greek = Vega
    BSValuesOption[i][j] = nu;
}
else if (iSigDeriv == 2)
{
    //greek = Volga
    BSValuesOption[i][j] = nu * d1 * d2 / sigma[volatility - 1];
}
else if (iSDeriv == 1)
{
    //greek = delta
    if (iSigDeriv == 0)
    {
        BSValuesOption[i][j] =
            Math.Exp((-dividend) * tau) *
            Cdf.Normal(d1);
    }
    else if (iSigDeriv == 1)
    {
        //greek = Vanna
        BSValuesOption[i][j] = (nu / S) *
            (1.0 - d1 / sqrt_sigsqtau);
    }
}
else if (iSDeriv == 2)
{
    if (ispeed == 0)
    {
        //greek = gamma
        BSValuesOption[i][j] =
            Math.Exp((-dividend) * tau) *
            n01pdf_d1 /(S * sqrt_sigsqtau);
    }
    else
    {
        //greek = speed
        BSValuesOption[i][j] =
            (-Math.Exp((-dividend) * tau)) *
            n01pdf_d1 * (1.0 + d1 / sqrt_sigsqtau) /
            (S * S * sqrt_sigsqtau);
    }
}
else if (iSDeriv == 3 && isspeed == 2)
{
    //greek = speed
    BSValuesOption[i][j] =
        (-Math.Exp((-dividend) * tau)) *
        n01pdf_d1 * (1.0 + d1 / sqrt_sigsqtau) /
        (S * S * sqrt_sigsqtau);
}
else if (iTDeriv == 1)
double d2 = d1 - sqrt_sigsqtau;
if (iSDeriv == 0)
{
    //greek = theta
    BSValuesOption[i][j] =
        Math.Exp((-dividend) * tau) * S *
        (dividend * Cdf.Normal(d1) - 0.5 * sigsq *
         n01pdf_d1 / sqrt_sigsqtau) - r *
        Math.Exp((-r) * tau) *
        strikePrices[strikePrice - 1] *
        Cdf.Normal(d2);
} else if (iSDeriv == 1)
{
    //greek = charm
    BSValuesOption[i][j] =
        Math.Exp((-dividend) * tau) * ((-dividend) * Cdf.Normal(d1) + n01pdf_d1 *
         ((r - dividend) * tau - 0.5 * d2 *
          sqrt_sigsqtau) / (tau * sqrt_sigsqtau));
} else if (iSDeriv == 2)
{
    //greek = color
    BSValuesOption[i][j] =
        (-Math.Exp((-dividend) * tau)) * n01pdf_d1 *
        (2.0 * dividend * tau + 1.0 + d1 *
         (2.0 * (r - dividend) * tau - d2 *
          sqrt_sigsqtau) / sqrt_sigsqtau)) /
        (2.0 * S * tau * sqrt_sigsqtau);
}
}

double relerrmax = 0.0;
int gNi = 3 * iTDeriv + iSDeriv;
if (iSigDeriv == 1 && iSDeriv == 0)
gNi = 6; //vega
if (iSigDeriv == 2 && iSDeriv == 0)
gNi = 7; //volga
if (iSigDeriv == 1 && iSDeriv == 1)
gNi = 8; //vanna
if (iRDeriv == 1)
gNi = 9; //rho
if (ispeed >= 1)
gNi = 9 + ispeed; //speed
for (int i = 0; i < nv; i++)
{
    for (int j = 0; j < nt; j++)
    {
        double sVo = splineValuesOption[j][i];
        double BSVo = BSValuesOption[j][i];
        //greeks(itd=1) ~ d/dtau = ~d/dt for iSDeriv > 0:
if (iTDeriv == 1 && iSDeriv > 0)
    sVo = -sVo;
    double relerr = Math.Abs((sVo - BSVo) / BSVo);
    if (relerr > relerrmax)
        relerrmax = relerr;
    reavg[gNi] += relerr;
    irect[gNi] += 1;
}
if (relerrmax > rex[gNi])
    rex[gNi] = relerrmax;
if ((((volatility == 1) && (strikePrice == 1))
{
    for (int i = 0; i < nv; i++)
    {
        Console.Out.Write("underlying price: {0,4:f1}; FK " +
        greekName[gNi] + ": ", evaluationPoints[i]);
        for (int j = 0; j < nt; j++)
        {
            double sVo = splineValuesOption[j][i];
            //greeks(itd=1) ~ d/dtau = -d/dt for isd > 0:
            if (iTDeriv == 1 && iSDeriv > 0)
                sVo = -sVo;
            Console.Out.Write("{0,13:f10} ", sVo);
        }
        Console.Out.WriteLine();
        Console.Out.Write(" BS " +
        greekName[gNi] + ": ");
        for (int j = 0; j < nt; j++)
        {
            Console.Out.Write("{0,13:f10} ",
            BSVvaluesOption[j][i]);
        }
        Console.Out.WriteLine();
    }
}
}
}

class MyCoefficients : FeynmanKac.IPdeCoefficients
{
    const double zero = 0.0;
    const double half = 0.5;
    double sigma, strikePrice, interestRate;
    double alpha, dividend;

    public MyCoefficients(double[] myData)
    {
        this.strikePrice = myData[0];
        this.sigma = myData[2];
    }
this.alpha = myData[3];
this.interestRate = myData[4];
this.dividend = myData[5];
}

// The coefficient sigma(x)
public double Sigma(double x, double t)
{
    return (sigma * Math.Pow(x, alpha * half));
}

// The coefficient derivative d(sigma) / dx
public double SigmaPrime(double x, double t)
{
    return (half * alpha * sigma * Math.Pow(x, alpha * half - 1.0));
}

// The coefficient mu(x)
public double Mu(double x, double t)
{
    return ((interestRate - dividend) * x);
}

// The coefficient kappa(x)
public double Kappa(double x, double t)
{
    return interestRate;
}

class MyBoundaries : FeynmanKac.IBoundaries
{

double xMax, df, interestRate, strikePrice;

public MyBoundaries(double[] myData)
{
    this.strikePrice = myData[0];
    this.xMax = myData[1];
    this.interestRate = myData[4];
}

public void LeftBoundaries(double time, double[,] bndCoeffs)
{
    bndCoeffs[0, 0] = 1.0;
    bndCoeffs[0, 1] = 0.0;
    bndCoeffs[0, 2] = 0.0;
    bndCoeffs[0, 3] = 0.0;
    bndCoeffs[1, 0] = 0.0;
    bndCoeffs[1, 1] = 1.0;
    bndCoeffs[1, 2] = 0.0;
    bndCoeffs[1, 3] = 0.0;
    bndCoeffs[2, 0] = 0.0;
    bndCoeffs[2, 1] = 0.0;
    bndCoeffs[2, 2] = 1.0;
    bndCoeffs[2, 3] = 0.0;
return;
}

public void RightBoundaries(double time, double[,] bndCoeffs)
{
    df = Math.Exp(interestRate * time);
    bndCoeffs[0, 0] = 1.0;
    bndCoeffs[0, 1] = 0.0;
    bndCoeffs[0, 2] = 0.0;
    bndCoeffs[0, 3] = xMax - df * strikePrice;
    bndCoeffs[1, 0] = 0.0;
    bndCoeffs[1, 1] = 1.0;
    bndCoeffs[1, 2] = 0.0;
    bndCoeffs[1, 3] = 1.0;
    bndCoeffs[2, 0] = 0.0;
    bndCoeffs[2, 1] = 0.0;
    bndCoeffs[2, 2] = 1.0;
    bndCoeffs[2, 3] = 0.0;

    return;
}

public double Terminal(double x)
{
    double zero = 0.0;
    // The payoff function
    double val = Math.Max(x - strikePrice, zero);
    return val;
}

Output

Constant Elasticity of Variance Model for Vanilla Call Option

Interest Rate: 0.050  Continuous Dividend: 0.000
Minimum and Maximum Prices of Underlying: 0.00 60.00
Number of equally spaced spline knots: 120
Number of unknowns: 363

Time in Years Prior to Expiration: 0.0833 0.3333 0.5833
Option valued at Underlying Prices: 19.00 20.00 21.00

3 point (parabolic) estimate of parameter derivatives;
espsfrac = 0.00100000

Strike=15.00, Sigma= 0.20, Alpha= 2.00:

yours to expiration: 0.0833 0.3333 0.5833

underlying price: 19.0; FK Value: 4.0623732450 4.2575924184 4.4733805278
    BS Value: 4.0623732509 4.2575929678 4.4733814062
underlying price: 20.0; FK Value: 5.0623700127 5.2505145764 5.4492418798
    BS Value: 5.0623700120 5.2505143129 5.4492428547
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>21.0</td>
<td>6.0623699727</td>
<td>6.2485587059</td>
<td>6.4385718831</td>
<td>6.0623699726</td>
<td>6.2485585270</td>
<td>6.4385706888</td>
<td>1.2448021908</td>
<td>4.8929216544</td>
<td>8.3041708173</td>
<td>1.2448024992</td>
<td>4.9107294561</td>
<td>8.4114990380</td>
<td>0.0003298970</td>
<td>0.3478163323</td>
<td>0.3076974472</td>
<td>0.0003295819</td>
<td>0.3487355010</td>
<td>0.3076857196</td>
<td>0.0000056652</td>
<td>0.1224632724</td>
</tr>
<tr>
<td>19.0</td>
<td>6.0623699727</td>
<td>6.2485587059</td>
<td>6.4385718831</td>
<td>1.2448021908</td>
<td>4.8929216544</td>
<td>8.3041708173</td>
<td>1.2448024992</td>
<td>4.9107294561</td>
<td>8.4114990380</td>
<td>0.0003298970</td>
<td>0.3478163323</td>
<td>0.3076974472</td>
<td>0.0003295819</td>
<td>0.3487355010</td>
<td>0.3076857196</td>
<td>0.0000056652</td>
<td>0.1224632724</td>
<td>0.6032458218</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.0</td>
<td>6.0623699727</td>
<td>6.2485587059</td>
<td>6.4385718831</td>
<td>1.2448021908</td>
<td>4.8929216544</td>
<td>8.3041708173</td>
<td>1.2448024992</td>
<td>4.9107294561</td>
<td>8.4114990380</td>
<td>0.0003298970</td>
<td>0.3478163323</td>
<td>0.3076974472</td>
<td>0.0003295819</td>
<td>0.3487355010</td>
<td>0.3076857196</td>
<td>0.0000056652</td>
<td>0.1224632724</td>
<td>0.6032458218</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21.0</td>
<td>6.0623699727</td>
<td>6.2485587059</td>
<td>6.4385718831</td>
<td>1.2448021908</td>
<td>4.8929216544</td>
<td>8.3041708173</td>
<td>1.2448024992</td>
<td>4.9107294561</td>
<td>8.4114990380</td>
<td>0.0003298970</td>
<td>0.3478163323</td>
<td>0.3076974472</td>
<td>0.0003295819</td>
<td>0.3487355010</td>
<td>0.3076857196</td>
<td>0.0000056652</td>
<td>0.1224632724</td>
<td>0.6032458218</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

296 • FeynmanKac

IMSL C# Numerical Library
underlying price: 19.0; FK Charm: -0.0014382828 -0.0879903285 -0.0843323992
BS Charm: -0.0014397520 -0.0879913927 -0.0843403333
underlying price: 20.0; FK Charm: -0.0000284881 -0.0364107814 -0.0547260337
BS Charm: -0.0000285354 -0.0364209077 -0.0547074804
underlying price: 21.0; FK Charm: -0.0000003396 -0.0126436426 -0.0313343015
BS Charm: -0.0000003190 -0.0126437838 -0.0313252716
underlying price: 19.0; FK Color: 0.0051622176 0.0685064195 0.0299871130
BS Color: 0.0051777484 0.0684737183 0.0300398444
underlying price: 20.0; FK Color: 0.0001188761 0.0355826975 0.0274292189
BS Color: 0.0001205713 0.0355891884 0.0274307898
underlying price: 21.0; FK Color: 0.0000015431 0.0143174419 0.0190897160
BS Color: 0.0000015141 0.0143247729 0.0190752019

Greek: Value; avg rel err: 0.000146170676; max rel err: 0.009030693731
Greek: Delta; avg rel err: 0.000035817236; max rel err: 0.001158483024
Greek: Gamma; avg rel err: 0.001088309906; max rel err: 0.044839095787
Greek: Theta; avg rel err: 0.000054196357; max rel err: 0.001412847204
Greek: Charm; avg rel err: 0.001213365580; max rel err: 0.064577964461
Greek: Color; avg rel err: 0.003323775096; max rel err: 0.13635625079
Greek: Vega; avg rel err: 0.001512805322; max rel err: 0.106097327138
Greek: Volga; avg rel err: 0.058536091480; max rel err: 1.639568432709
Greek: Vanna; avg rel err: 0.0001061842143; max rel err: 0.065659421418
Greek: Rho; avg rel err: 0.000146868204; max rel err: 0.009438785575
Greek: Speed; avg rel err: 0.007252489626; max rel err: 0.12363047780
Greek: Speed2; avg rel err: 0.008430324710; max rel err: 0.255782408454

FeynmanKac.PDEStepControlMethod Enumeration

public enumeration Imsl.Math.FeynmanKac.PDEStepControlMethod

Indicates which step control method is used in the integration of the Feynman-Kac PDE.

Fields

MethodOfPetzold

public Imsl.Math.FeynmanKac.PDEStepControlMethod MethodOfPetzold

Description

Indicates that the step control algorithm of the original Petzold code is used in the integration.

MethodOfSoederlind

public Imsl.Math.FeynmanKac.PDEStepControlMethod MethodOfSoederlind
Description
Indicates that the step control method by Soederlind is used in the integration.

FeynmanKac.IPdeCoefficients Interface

public interface Imsl.Math.FeynmanKac.IPdeCoefficients
Interface for computation of coefficients of the Feynman-Kac PDE.

Methods

Kappa
abstract public double Kappa(double x, double t)

Description
Returns the value of the $\kappa$ coefficient at the given point.

Parameters
- x – A double, the point in space at which $\kappa$ is to be evaluated.
- t – A double, the time point at which $\kappa$ is to be evaluated.

Returns
A double, the value of $\kappa$ at $(x,t)$.

Remarks
Time dependency of $\kappa$ can be controlled via method SetTimeDependence. Use of this method will usually yield a more efficient algorithm because some finite element matrices do not have to be reassembled for each $t$ value.

Mu
abstract public double Mu(double x, double t)

Description
Returns the value of the $\mu$ coefficient at the given point.

Parameters
- x – A double, the point in space at which $\mu$ is to be evaluated.
- t – A double, the time point at which $\mu$ is to be evaluated.

Returns
A double, the value of $\mu$ at $(x,t)$. 
Remarks
Time dependency of \( \mu \) can be controlled via method `SetTimeDependence`. Use of this method will usually yield a more efficient algorithm because some finite element matrices do not have to be reassembled for each \( t \) value.

**Sigma**
abstract public double Sigma(double x, double t)

**Description**
Returns the value of the \( \sigma \) coefficient at the given point.

**Parameters**
- \( x \) – A double scalar, the point in space at which \( \sigma \) is to be evaluated.
- \( t \) – A double scalar, the time point at which \( \sigma \) is to be evaluated.

**Returns**
A double scalar, the value of \( \sigma \) at \((x,t)\).

**Remarks**
Time dependency of \( \sigma \) can be controlled via method `SetTimeDependence`. Use of this method will usually yield a more efficient algorithm because some finite element matrices do not have to be reassembled for each \( t \) value.

**SigmaPrime**
abstract public double SigmaPrime(double x, double t)

**Description**
Returns the value of \( \sigma' = \frac{\partial \sigma(x,t)}{\partial x} \) at the given point.

**Parameters**
- \( x \) – A double, the point in space at which \( \sigma' \) is to be evaluated.
- \( t \) – A double, the time point at which \( \sigma' \) is to be evaluated.

**Returns**
A double, the value of \( \sigma' \) at \((x,t)\).

**Remarks**
Time dependency of \( \sigma' \) can be controlled via method `SetTimeDependence`. Use of this method will usually yield a more efficient algorithm because some finite element matrices do not have to be reassembled for each \( t \) value.

---

**FeynmanKac.IBoundaries Interface**

public interface Imsl.Math.FeynmanKac.IBoundaries

Public interface for user supplied boundary coefficients and terminal condition the PDE must satisfy.
Methods

LeftBoundaries
abstract public void LeftBoundaries(double time, double[,] coefficients)

Description
Returns the coefficient values of the left boundary conditions. There are numLeftBounds conditions specified at the left end, \( x_{\text{min}} \). The left boundary conditions are
\[
a_i(x,t)f + b_i(x,t)f_x + c_i(x,t)f_{xx} = d_i(x,t), \quad x = x_{\text{min}}, \ 1 \leq i \leq \text{numLeftBounds}.
\]

Parameters
- **time** – A double, the time point at which the boundary coefficients are to be evaluated.
- **coefficients** – An output double matrix of dimension numLeftBounds by 4 containing the computed boundary coefficient values. The coefficients are stored row-wise according to the matrix scheme
\[
(a_i(x_{\text{min}},t), b_i(x_{\text{min}},t), c_i(x_{\text{min}},t), d_i(x_{\text{min}},t))_{1 \leq i \leq \text{numLeftBounds}}.
\]

Remarks
Time dependency of the boundary coefficients can be controlled via method SetTimeDependence. Use of this method will usually yield a more efficient algorithm because some finite element matrices do not have to be reassembled for each \( t \) value.

RightBoundaries
abstract public void RightBoundaries(double time, double[,] coefficients)

Description
Returns the coefficient values of the right boundary conditions. There are numRightBounds conditions specified at the right end, \( x_{\text{max}} \). The right boundary conditions are
\[
a_i(x,t)f + b_i(x,t)f_x + c_i(x,t)f_{xx} = d_i(x,t), \quad x = x_{\text{max}}, \ 1 \leq i \leq \text{numRightBounds}.
\]

Parameters
- **time** – A double, the time point at which the boundary coefficients are to be evaluated.
- **coefficients** – An output double matrix of dimension numRightBounds by 4, containing the computed boundary coefficient values. The coefficients are stored row-wise according to the matrix scheme
\[
(a_i(x_{\text{max}},t), b_i(x_{\text{max}},t), c_i(x_{\text{max}},t), d_i(x_{\text{max}},t))_{1 \leq i \leq \text{numRightBounds}}.
\]

Remarks
Time dependency of the boundary coefficients can be controlled via method SetTimeDependence. Use of this method will usually yield a more efficient algorithm because some finite element matrices do not have to be reassembled for each \( t \) value.

Terminal
abstract public double Terminal(double z)
Description
Returns the terminal condition value.

Parameter
z – A double scalar, the point in x-direction, where the terminal condition is evaluated.

Returns
A double scalar, the value of the terminal condition \( p(x) \) at point \( z \).

FeynmanKac.IInitialData Interface

public interface Imsl.Math.FeynmanKac.IInitialData
Public interface for adjustment of initial data or as an opportunity for output during the integration steps.

Method

Init
abstract public void Init(double[] xGrid, double[] tGrid, double time, double[] yprime, double[] y, double[] absoluteErrorTolerance, double[] relativeErrorTolerance)

Description
Method that allows for adjustment of initial data or as an opportunity for output during the integration steps.

Parameters
xGrid – A double array containing the grid points in the x-direction.
tGrid – A double array containing the grid points in the t-direction.
time – A double scalar containing the time point for the evaluation. Possible values are 0 (the initial or “terminal” time point) and all values in array tGrid.
yprime – An input double array of length 3*xGrid.Length containing the derivatives of the Hermite quintic spline coefficients at time point time.
y – A double array of length 3*xGrid.Length containing the coefficients of the Hermite quintic spline at time point time. For the initial time point time = 0 this array can be used to reset the Hermite quintic spline coefficients to user defined values. For all other values of time array y is an input array.
absoluteErrorTolerance – An input or output double array of length 3*xGrid.Length containing absolute error tolerances.
relativeErrorTolerance – An input or output double array of length 3*xGrid.Length containing relative error tolerances.
FeynmanKac.IForcingTerm Interface

```java
public interface Imsl.Math.FeynmanKac.IForcingTerm

Public interface for non-zero forcing term in the Feynman-Kac equation.

Method

Force

abstract public void Force(int interval, double[] y, double time, double width,
  double[] xlocal, double[] qw, double[,] u, double[] phi, double[,] dphi)

Description

Computes approximations to the forcing term \( \phi(f, x, t) \) and its derivative \( \partial \phi / \partial y \).

Parameters

- **interval** – An int, the index related to the integration interval \([xGrid[interval-1], xGrid[interval]]\).
- **y** – An input double array of length 3*xGrid.Length containing the coefficients of the Hermite quintic spline representing the solution of the Feynman-Kac equation at time point \( t \). For each \( x \in [x_i, x_{i+1}], h_i = x_{i+1} - x_i, z_i = (x - x_i) / h_i, i = 1, \ldots, xGrid.Length - 1 \) the approximate solution is locally defined by
  \[
  f(x, t) = f_i b_0(z) + f_{i+1} b_0(1 - z) + h_i f'_i b_1(z) - h_i f'_{i+1} b_1(1 - z) + h_i^2 f''_i b_2(z) + h_i^2 f''_{i+1} b_2(1 - z).
  \]
  The values \( y_{3i-2} = f_i, y_{3i-1} = f'_i, y_{3i} = f''_i, i = 1, \ldots, xGrid.Length \), are stored as successive triplets in \( y \).
- **time** – A double scalar, the time point.
- **width** – A double scalar, the width of the integration interval, \( width=xGrid[interval]-xGrid[interval-1] \).
- **xlocal** – An input double array containing the Gauss-Legendre points translated and normalized to the interval \([xGrid[interval-1], xGrid[interval]]\). The array length is equal to the degree of the Gauss-Legendre polynomial.
- **qw** – An input double array containing the Gauss-Legendre weights. The array length is equal to the degree of the Gauss-Legendre polynomial.
- **u** – An input double matrix of dimension 12 by \( xlocal.Length \) containing the basis function values that define \( \tilde{\beta}(x) \) at the Gauss-Legendre points \( xlocal \). Setting
  \[
  u_{k,i} := u[k,i] \quad \text{and} \quad x_i := xlocal[i],
  \]
The vector $\tilde{\beta}(x_i)$ is defined as

$$\tilde{\beta}(x_i) := (\beta_{3\text{-(interval-1)}}(x_i), \ldots, \beta_{3\text{+interval+2}}(x_i))^T = (u_{0,i}, u_{1,i}, u_{2,i}, u_{6,i}, u_{7,i}, u_{8,i})^T.$$ 

$\phi$ – An output double array of length 6 containing Gauss-Legendre approximations for the local contributions

$$\phi := \int_{xgrid[\text{interval-1}]}^{xgrid[\text{interval}]} \phi(f, x, t) \tilde{\beta}(x) dx,$$

where $t=\text{time}$ and $\tilde{\beta}(x) := (\beta_{3\text{-(interval-1)}}(x), \ldots, \beta_{3\text{+interval+2}}(x))^T$. Denoting by degree the number of Gauss-Legendre points ($xlocal.Length$) and setting $x_j := xlocal[j]$, vector phi contains elements

$$\phi[i] = \text{width} \times \sum_{j=0}^{\text{degree-1}} qw[j] \tilde{\beta}_i(x_j) \phi(f, x_j, t)$$

for $i=0, \ldots, 5$.

dphi – An output double matrix of dimension 6 by 6 containing a Gauss-Legendre approximation for the Jacobian of the local contributions $\phi$ at time point $t=\text{time}$,

$$\frac{\partial \phi}{\partial y} = \int_{xgrid[\text{interval-1}]}^{xgrid[\text{interval}]} \frac{\partial \phi(f, x, t)}{\partial f} \tilde{\beta}(x) \tilde{\beta}^T(x) dx.$$ 

The approximation to this symmetric matrix is stored row-wise, i.e.

$$dphi[i,j] = \text{width} \times \sum_{k=0}^{\text{degree-1}} qw[k] \tilde{\beta}_i(x_k) \tilde{\beta}_j(x_k) \frac{\partial \phi}{\partial f} \bigg|_{x=xlocal[k], t=\text{time}}$$

for $i,j=0, \ldots, 5$. 

Chapter 5. Differential Equations

FeynmanKac.IForcingTerm • 303
Chapter 6: Transforms

Types

class FFT ................................................................. 306
class ComplexFFT ...................................................... 310

Usage Notes

Fast Fourier Transforms

A fast Fourier transform (FFT) is simply a discrete Fourier transform that is computed efficiently. Basically, the straightforward method for computing the Fourier transform takes approximately $n^2$ operations where $n$ is the number of points in the transform, while the FFT (which computes the same values) takes approximately $n \log n$ operations. The algorithms in this chapter are modeled on the Cooley-Tukey (1965) algorithm. Hence, these functions are most efficient for integers that are highly composite; that is, integers that are a product of small primes.

For the two classes, FFT and ComplexFFT, a single instance can be used to transform multiple sequences of the same length. In this situation, the constructor computes the initial setup once. This may result in substantial computational savings. For more information on the use of these classes consult the documentation under the appropriate class name.

Continuous Versus Discrete Fourier Transform

There is, of course, a close connection between the discrete Fourier transform and the continuous Fourier transform. Recall that the continuous Fourier transform is defined (Brigham 1974) as

\[
\hat{f}(\omega) = (\mathcal{F}f)(\omega) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \omega t} dt
\]

We begin by making the following approximation:

\[
\hat{f}(\omega) \approx \int_{-T/2}^{T/2} f(t) e^{-2\pi i \omega t} dt
\]
\begin{align*}
\int_0^T f(t - T/2)e^{-2\pi i\omega(t - T/2)}dt &= e^{\pi i\omega T} \int_0^T f(t - T/2)e^{-2\pi i\omega t}dt \\
\end{align*}

If we approximate the last integral using the rectangle rule with spacing \( h = T/n \), we have

\[ \hat{f}(\omega) \approx e^{\pi i\omega T} h \sum_{k=0}^{n-1} e^{-2\pi i\omega k} f(kh - T/2) \]

Finally, setting \( \omega = j/T \) for \( j = 0, \ldots, n-1 \) yields

\[ \hat{f}(j/T) \approx e^{\pi ij} h \sum_{k=0}^{n-1} e^{-2\pi ijk/n} f(kh - T/2) = (-1)^j \sum_{k=0}^{n-1} e^{-2\pi ijk/n} f_k \]

where the vector \( f^h = (f(-T/2), \ldots, f((n-1)h-T/2)) \). Thus, after scaling the components by \((-1)^h\), the discrete Fourier transform, as computed in ComplexFFT (with input \( f^h \)) is related to an approximation of the continuous Fourier transform by the above formula.

### FFT Class

#### public class Imsl.Math.FFT

FFT functions.

Class FFT computes the discrete Fourier transform of a real vector of size \( n \). The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when \( n \) is a product of small prime factors. If \( n \) satisfies this condition, then the computational effort is proportional to \( n \log n \).

The Forward method computes the forward transform. If \( n \) is even, then the forward transform is

\[
q_{2m-1} = \sum_{k=0}^{n-1} p_k \cos \frac{2\pi km}{n} \quad m = 1, \ldots, n/2
\]

\[
q_{2m-2} = -\sum_{k=0}^{n-1} p_k \sin \frac{2\pi km}{n} \quad m = 1, \ldots, n/2 - 1
\]
If \( n \) is odd, \( q_m \) is defined as above for \( m \) from 1 to \((n - 1)/2\).

Let \( f \) be a real valued function of time. Suppose we sample \( f \) at \( n \) equally spaced time intervals of length \( \delta \) seconds starting at time \( t_0 \). That is, we have

\[
p_i := f(t_0 + i\Delta) \quad i = 0, 1, \ldots, n - 1
\]

We will assume that \( n \) is odd for the remainder of this discussion. The class FFT treats this sequence as if it were periodic of period \( n \). In particular, it assumes that \( f(t_0) = f(t_0 + n\Delta) \). Hence, the period of the function is assumed to be \( T = n\Delta \). We can invert the above transform for \( p \) as follows:

\[
p_m = \frac{1}{n} \left[ q_0 + 2 \sum_{k=0}^{(n-3)/2} q_{2k+1} \cos \left( \frac{2\pi(k + 1)m}{n} \right) - 2 \sum_{k=0}^{(n-3)/2} q_{2k+2} \sin \left( \frac{2\pi(k + 1)m}{n} \right) \right]
\]

This formula is very revealing. It can be interpreted in the following manner. The coefficients \( q \) produced by FFT determine an interpolating trigonometric polynomial to the data. That is, if we define

\[
g(t) = \frac{1}{n} \left[ q_0 + 2 \sum_{k=0}^{(n-3)/2} q_{2k+1} \cos \left( \frac{2\pi(k + 1)(t - t_0)}{n\Delta} \right) - 2 \sum_{k=0}^{(n-3)/2} q_{2k+2} \sin \left( \frac{2\pi(k + 1)(t - t_0)}{n\Delta} \right) \right]
\]

\[
= \frac{1}{n} \left[ q_0 + 2 \sum_{k=0}^{(n-3)/2} q_{2k+1} \cos \left( \frac{2\pi(k + 1)(t - t_0)}{T} \right) - 2 \sum_{k=0}^{(n-3)/2} q_{2k+2} \sin \left( \frac{2\pi(k + 1)(t - t_0)}{T} \right) \right]
\]

then we have

\[
f(t_0 + (i - 1)\Delta) = g(t_0 + (i - 1))\Delta
\]

Now suppose we want to discover the dominant frequencies, forming the vector \( P \) of length \((n + 1)/2\) as follows:

\[
P_0 := |q_0|
\]
\[ P_k := \sqrt{q_{2k-2}^2 + q_{2k-1}^2} \quad k = 1, 2, \ldots, (n-1)/2 \]

These numbers correspond to the energy in the spectrum of the signal. In particular, \( P_k \) corresponds to the energy level at frequency

\[ \frac{k}{T} = \frac{k}{n\Delta} \quad k = 0, 1, \ldots, \frac{n-1}{2} \]

Furthermore, note that there are only \((n+1)/2 \approx T/(2\Delta)\) resolvable frequencies when \( n \) observations are taken. This is related to the Nyquist phenomenon, which is induced by discrete sampling of a continuous signal. Similar relations hold for the case when \( n \) is even.

If the backward method is used, then the backward transform is computed. If \( n \) is even, then the backward transform is

\[
q_m = p_0 + (-1)^m p_{n-1} + 2 \sum_{k=0}^{n/2-1} p_{2k+1} \cos \frac{2\pi (k+1)m}{n} - 2 \sum_{k=0}^{n/2-2} p_{2k+2} \sin \frac{2\pi (k+1)m}{n}
\]

If \( n \) is odd,

\[
q_m = p_0 + 2 \sum_{k=0}^{(n-3)/2} p_{2k+1} \cos \frac{2\pi (k+1)m}{n} - 2 \sum_{k=0}^{(n-3)/2} p_{2k+2} \sin \frac{2\pi (k+1)m}{n}
\]

The backward Fourier transform is the unnormalized inverse of the forward Fourier transform.

FFT is based on the real FFT in FFTPACK, which was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

**Constructor**

**FFT**

```csharp
public FFT(int n)
```

**Description**

Constructs an FFT object.

**Parameter**

- \( n \) – A int which specifies the array size that this object can handle.
Methods

**Backward**

```java
public double[] Backward(double[] coef)
```

**Description**

Compute the real periodic sequence from its Fourier coefficients.

**Parameter**

- `coef` – A `double` array containing the Fourier coefficients.

**Returns**

A `double` array containing the periodic sequence.

**Forward**

```java
public double[] Forward(double[] seq)
```

**Description**

Compute the Fourier coefficients of a real periodic sequence.

**Parameter**

- `seq` – A `double` array containing the sequence to be transformed.

**Returns**

A `double` array containing the transformed sequence.

**Example: Fast Fourier Transform**

The Fourier coefficients of a periodic sequence are computed. The coefficients are then used to reproduce the periodic sequence.

```java
using System;
using Imsl.Math;

public class FFTEx1
{
    public static void Main(string[] args)
    {
        double[] x = new double[]{1, 2, 3, 4, 5, 6, 7, 8};
        FFT fft = new FFT(x.Length);

        double[] y = fft.Forward(x);
        double[] z = fft.Backward(y);
        for (int i = 0; i < x.Length; i++)
        {
            z[i] = z[i] / x.Length;
        }

        new PrintMatrix("x").Print(x);
    }
}
```

Transforms
new PrintMatrix("y").Print(y);
new PrintMatrix("z").Print(z);
}
}

**Output**

\[
x
0
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 8
\]

\[
y
0 36
1 -4
2 9.65685424949238
3 -4
4 4
5 -4
6 1.65685424949238
7 -4
\]

\[
z
0
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 8
\]

---

**ComplexFFT Class**

```csharp
public class Imsl.Math.ComplexFFT

Complex FFT.

Class ComplexFFT computes the discrete complex Fourier transform of a complex vector of size \( N \). The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when \( N \) is a product of
```
small prime factors. If $N$ satisfies this condition, then the computational effort is proportional to $N \log N$. This considerable savings has historically led people to refer to this algorithm as the “fast Fourier transform” or FFT.

Specifically, given an $N$-vector $x$, method Forward returns

$$c_m = \sum_{n=0}^{N-1} x_n e^{-2\pi inm/N}$$

Furthermore, a vector of Euclidean norm $S$ is mapped into a vector of norm

$$\sqrt{NS}$$

Finally, note that we can invert the Fourier transform as follows:

$$x_n = \frac{1}{N} \sum_{j=0}^{N-1} c_m e^{2\pi inj/N}$$

This formula reveals the fact that, after properly normalizing the Fourier coefficients, one has the coefficients for a trigonometric interpolating polynomial to the data. An unnormalized inverse is implemented in Backward. ComplexFFT is based on the complex FFT in FFTPACK. The package, FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Specifically, given an $N$-vector $c$, Backward returns

$$s_m = \sum_{n=0}^{N-1} c_n e^{2\pi inm/N}$$

Furthermore, a vector of Euclidean norm $S$ is mapped into a vector of norm

$$\sqrt{NS}$$

Finally, note that we can invert the inverse Fourier transform as follows:

$$c_n = \frac{1}{N} \sum_{m=0}^{N-1} s_m e^{-2\pi inm/N}$$

This formula reveals the fact that, after properly normalizing the Fourier coefficients, one has the coefficients for a trigonometric interpolating polynomial to the data. Backward is based on the complex inverse FFT in FFTPACK. The package, FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.
Constructor

ComplexFFT
public ComplexFFT(int n)

Description
Constructs a complex FFT object.

Parameter
n – A int which specifies the array size that this object can handle.

Methods

Backward

Description
Compute the complex periodic sequence from its Fourier coefficients.

Parameter
coef – A Complex array of Fourier coefficients.

Returns
A Complex array containing the periodic sequence.

Forward

Description
Compute the Fourier coefficients of a complex periodic sequence.

Parameter
seq – A Complex array containing the sequence to be transformed.

Returns
A Complex array containing the transformed sequence.

Example: Complex FFT
The Fourier coefficients of a complex periodic sequence are computed. Then the coefficients are used to try to reproduce the periodic sequence.

using System;
using Imsl.Math;
public class ComplexFFTEx1
{
    public static void Main(String[] args)
    {
        Complex[] x = new Complex[]{
            new Complex(1, 8), new Complex(2, 7), new Complex(3, 6),
            new Complex(4, 5), new Complex(5, 4), new Complex(6, 3),
            new Complex(7, 2), new Complex(8, 1)
        };
        ComplexFFT fft = new ComplexFFT(x.Length);
        Complex[] y = fft.Forward(x);
        Complex[] z = fft.Backward(y);
        for (int i = 0; i < x.Length; i++)
        {
            z[i] /= x.Length;
        }
        new PrintMatrix("x").Print(x);
        new PrintMatrix("y").Print(y);
        new PrintMatrix("z").Print(z);
    }
}

Output

 x
 0 0
 1 1+8i
 2 2+7i
 3 3+6i
 4 4+5i
 5 5+4i
 6 6+3i
 7 7+2i
 8 8+1i

 y
 0 36+36i
 1 5.65685424949238+13.65685424949241i
 2 +8i
 3 -2.34314575050762+5.65685424949238i
 4 -4+4i
 5 -5.65685424949238+2.34314575050762i
 6 -8
 7 -13.6568542494924-5.65685424949238i

 z
 0 1+8i
 1 2+7i
 2 3+6i
 3 4+5i
 4 5+4i

Transforms
<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>6+3i</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>7+2i</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8+1i</td>
</tr>
</tbody>
</table>
Chapter 7: Nonlinear Equations

Types

class ZeroPolynomial ............................................................... 316
class ZerosFunction ............................................................. 320
interface ZerosFunction.IFunction ........................................... 326
class ZeroSystem ................................................................. 327
interface ZeroSystem.IFunction ............................................. 332
interface ZeroSystem.IJacobian ............................................. 333

Usage Notes

Zeros of a Polynomial

A polynomial function of degree $n$ can be expressed as follows:

$$ p(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0 $$

where $a_n \neq 0$. The ZeroPolynomial class finds zeros of a polynomial with real or complex coefficients using Aberth’s method.

Zeros of a Function

The ZerosFunction class uses Muller’s method to find the real zeros of a real-valued function.

Root of System of Equations

A system of equations can be stated as follows:

$$ f_i(x) = 0, \text{ for } i = 1, 2, \ldots, n $$

where $x \in \mathbb{R}^n$, and $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$. The ZeroSystem class uses a modified hybrid method due to M.J.D. Powell to find the zero of a system of nonlinear equations.
ZeroPolynomial Class

public class Imsl.Math.ZeroPolynomial

The ZeroPolynomial class computes the zeros of a polynomial with complex coefficients, Aberth’s method.

This class is a translation of a Fortran code written by Dario Andrea Bini, University of Pisa, Italy (bini@dm.unipi.it). Numerical computation of polynomial zeros by means of Aberth’s method, Numerical Algorithms, 13 (1996), pp. 179-200.

The original Fortran code includes the following notice.

All the software contained in this library is protected by copyright Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software.


Property

MaximumIterations

public int MaximumIterations {get; set; }

Description

The maximum number of iterations allowed.

Property Value

An int which specifies the maximum number of iterations allowed. The default value is 30.

Exception

System.ArgumentException is thrown if MaxIterations is less than or equal to zero
Constructor

ZeroPolynomial
public ZeroPolynomial()

Description
Creates an instance of the solver.

Methods

ComputeRoots
public Imsl.Math.Complex[] ComputeRoots(double[] coef)

Description
Computes the roots of the polynomial with real coefficients.

\[ p(x) = \text{coef}[n] \times x^n + \text{coef}[n-1] \times x^{n-1} + \ldots + \text{coef}[0] \]

Parameter
coef – A double array containing the polynomial coefficients.

Returns
A Complex array containing the roots of the polynomial.

Exception
Imsl.Math.DidNotConvergeException is thrown if the iteration did not converge.

ComputeRoots

Description
Computes the roots of the polynomial with Complex coefficients.

\[ p(x) = \text{coef}[n] \times x^n + \text{coef}[n-1] \times x^{n-1} + \ldots + \text{coef}[0] \]

Parameter
coef – A Complex array containing the polynomial coefficients.

Returns
A Complex array containing the roots of the polynomial.
Exception

Imsl.Math.DidNotConvergeException is thrown if if the iteration for the zeros did not converge.

GetRadius
public double GetRadius(int index)

Description
Returns an a-posteriori absolute error bound on the root.

Parameter

index – An int specifying the (0-based) index of the root whose error bound is to be returned.

Returns
A double representing the error bound on the index-th root.

Remarks
NaN is returned if the corresponding root cannot be represented as floating point due to overflow or underflow or if the roots have not yet been computed.

GetRoot
public Imsl.Math.Complex GetRoot(int index)

Description
Returns a zero of the polynomial.

Parameter

index – An int which specifies the (0-based) index of the root to be returned.

Returns
A Complex which represents the index-th root of the polynomial.

GetRoots
public Imsl.Math.Complex[] GetRoots()

Description
Returns the zeros of the polynomial.

Returns
A Complex array containing the roots of the polynomial.

GetStatus
public bool GetStatus(int index)

Description
Returns the error status of a root.

Parameter

index – An int representing the (0-based) index of the root whose error status is to be returned.
Returns
A bool representing the error status on the index-th root.

Remarks
It is false if the approximation of the index-th root has been carried out successfully, for example, the computed approximation can be viewed as the exact root of a slightly perturbed polynomial. It is true if more iterations are needed for the index-th root.

Example 1: Zeros of a Polynomial
The zeros of a polynomial with real coefficients are computed.

```csharp
using System;
using Imsl.Math;

public class ZeroPolynomialEx1
{
    public static void Main(String[] args)
    {
        double[] coef = new double[]{2, 4, -3, 1};
        ZeroPolynomial zp = new ZeroPolynomial();
        Complex[] root = zp.ComputeRoots(coef);
        for (int k = 0; k < root.Length; k++)
        {
            Console.Out.WriteLine("root = " + root[k].ToString);
            Console.Out.WriteLine(" radius = " + zp.GetRadius(k));
            Console.Out.WriteLine(" status = " + zp.GetStatus(k));
        }
    }
}
```

Output
```
root = 0.99999999999999978-0.99999999999999978i
radius = 1.99006775678924E-14
status = False
root = 1.0000000000000004+1.0000000000000002i
radius = 1.96185227616234E-14
status = False
root = 0.99999999999999989-1.6543612251060553E-24i
radius = 2.04503081135961E-14
status = False
```

Example 2: Zeros of a Polynomial with Complex Coefficients
The zeros of a polynomial with Complex coefficients are computed.

```csharp
using System;
using Imsl.Math;
```
public class ZeroPolynomialEx2
{
    public static void Main(String[] args)
    {
        // Find zeros of z^3-(3+6i)*z^2+(-8+12i)*z+10
        Complex[] coef = new Complex[]{
            new Complex(10),
            new Complex(-8, 12),
            new Complex(-3, -6),
            new Complex(1)};

        ZeroPolynomial zp = new ZeroPolynomial();
        Complex[] root = zp.ComputeRoots(coef);

        for (int k = 0; k < root.Length; k++)
        {
            Console.Out.WriteLine("root = "+ root[k]);
            Console.Out.WriteLine(" radius = "+
                zp.GetRadius(k).ToString("0.00e+0");
            Console.Out.WriteLine(" status = " + zp.GetStatus(k));
        }
    }
}

Output
root = 1.0000000000000004+1.0000000000000004i
  radius = 6.30e-14
  status = False
root = 0.99999999999999944+2.0000000000000018i
  radius = 1.96e-13
  status = False
root = 0.999999999999999+2.9999999999999987i
  radius = 1.47e-13
  status = False

ZerosFunction Class

public class Imsl.Math.ZerosFunction

Finds the real zeros of a real, continuous, univariate function, f(x).

ZerosFunction computes n real zeros of a real, continuous, univariate function f. The search for the zeros of the function can be limited to a specified interval, or extended over the entire real line. The algorithm is generally more efficient if an interval is specified. The user supplied function, f(x), must return valid results for all values in the specified interval. If no interval is given, the user-supplied function must return valid results for all real numbers.
The function has two convergence criteria. The first criterion accepts a root, \( x \), if
\[
|f(x)| \leq \tau
\]
where \( \tau = \text{Error} \), see property \( \text{Error} \).

The second criterion accepts a root if it is known to be inside of an interval of length at most \( \text{AbsoluteError} \), see property \( \text{AbsoluteError} \).

A root is accepted if it satisfies either criteria and is not within \( \text{MinimumSeparation} \) of another accepted root, see property \( \text{MinimumSeparation} \).

If initial guesses for the roots are given, Müller’s method (Müller 1956) is used for each of these guesses. For each guess, the Müller iteration is stopped if the next step would be outside of the bound, if given. The iteration is also stopped if the algorithm cannot make further progress in finding a root.

If no guesses for the zeros were given, or if Müller’s method with the guesses did not find the requested number of roots, a meta-algorithm, combining Müller’s and Brent’s methods, is used. Müller’s method is used primarily to find the roots of functions, such as \( f(x) = x^2 \), where the function does not cross the \( y=0 \) line. Brent’s method is used to find other types of roots.

The meta-algorithm successively refines the interval using a one-dimensional Faure low-discrepancy sequence. The Faure sequence may be scaled by setting the bound interval \([a,b]\) using the \texttt{SetBounds} method. The Faure sequence will be scaled from \((0,1)\) to \((a,b)\).

If no bound on the function’s domain is given, the entire real line must be searched for roots. In this case the Faure sequence is scaled from \((0, 1)\) to \((-\infty, +\infty)\) using the mapping
\[
h(u) = \text{xScale} \cdot \tan(\pi(u - 1/2))
\]
where \( \text{xScale} \) is set by the \texttt{xScale} property.

At each step of the iteration, the next point in the Faure sequence is added to the list of breakpoints defining the subintervals. Call the points \( x_0 = a, x_1 = b, x_2, x_3, \ldots \). The new point, \( x_s \), splits an existing subinterval, \([x_p, x_q]\).

The function is evaluated at \( x_s \). If its value is small enough, specifically if
\[
|f(x_s)| < \text{MullerTolerance}
\]
then Müller’s method is used with \( x_p, x_q \) and \( x_s \) as starting values. If a root is found, it is added to the list of roots. If more roots are required, the new Faure point is used.

If Müller’s method did not find a root using the new point, the function value at the point is compared with the function values at the endpoints of the subinterval it divides. If \( f(x_p) f(x_s) < 0 \) and no root has previously been found in \([x_p, x_s]\), then Brent’s method is used to find a root in this interval. Similarly, if the function changes sign over the interval \([x_s, x_q]\), and a root has not already been found in the subinterval, Brent’s method is used.
Properties

AbsoluteError
virtual public double AbsoluteError {get; set; }

Description
The second convergence criterion.

Property Value
A double value specifying the second convergence criterion. A root is accepted if the absolute value of the function at the point is less than or equal to AbsoluteError. AbsoluteError must be greater than or equal to 0.0.
Default: AbsoluteError = 2.22e-14

Remarks
The second criterion accepts a root if the root is known to be inside an interval of length at most AbsoluteError.

AllConverged
virtual public bool AllConverged {get; }

Description
Returns true if the iterations for all of the roots have converged.

Property Value
A bool representing status of the roots convergence. AllConverged = true if the iterations for all of the roots converged.

Error
virtual public double Error {get; set; }

Description
First convergence criterion.

Property Value
A double containing the first convergence criterion. Error must be greater than or equal to 0.0.
Default: Error = 2.0e-8/XScale.

Remarks
A root is accepted if it is bracketed within an interval of length Error, |f(x)| ≤ τ, where τ = Error.

MaxEvaluations
virtual public int MaxEvaluations {get; set; }

Description
The maximum number of function evaluations allowed.
Property Value
An int containing the maximum number of function evaluations allowed. Once this limit is reached, the roots found are returned. Default: MaxEvaluations = 100.

Remarks
Methods AllConverged and property NumberOfRootsFound can be used to confirm whether or not the number of roots requested were found within the maximum evaluations specified. MaxEvaluations must be greater than or equal to 0.

MinimumSeparation
virtual public double MinimumSeparation {get; set; }

Description
Sets the minimum separation between accepted roots.

Property Value
A double containing the minimum separation between accepted roots. If two points satisfy the convergence criteria, but are within MinimumSeparation of each other, only one of the roots is accepted. MinimumSeparation must be greater than or equal to 0.0. Default: MinimumSeparation = 1.0e-8/XScale.

MullerTolerance
virtual public double MullerTolerance {get; set; }

Description
Tolerance used during refinement to determine if Müller's method is started.

Property Value
A double containing the tolerance used during refinement to determine when the Müller's method is used. Default: MullerTolerance = 1.0e-8/AbsoluteError

Remarks
Müller's method is started if, during refinement, a point is found for which the absolute value of the function is less than MullerTolerance and the point is not near an already discovered root. If MullerTolerance is less than or equal to zero, Müller's method is never used.

NumberOfEvaluations
virtual public int NumberOfEvaluations {get; }

Description
The actual number of function evaluations performed.

Property Value
An int containing the actual number of function evaluations performed.

NumberOfRoots
virtual public int NumberOfRoots {get; set; }

Nonlinear Equations ZerosFunction • 323
**Description**  
The requested number of roots to be found.

**Property Value**  
An int containing the number of roots to be found. **NumberOfRoots** must be greater than or equal to zero.  
Default: **NumberOfRoots** = 1.

**NumberOfRootsFound**  
virtual public int NumberOfRootsFound {get; }

**Description**  
Returns the number of zeros found.

**XScale**  
virtual public double XScale {get; set; }

**Description**  
Sets the scaling in the x-coordinate.

**Property Value**  
A double containing the scaling in the x-coordinate. The absolute value of the roots divided by **XScale** should be about one. **XScale** must be greater than 0.0.  
Default: **XScale** = 1.0.

**Remarks**  
If no bound on the function’s domain is given, the entire real line must be searched for roots. In this case the Faure sequence is scaled from (0, 1) to \((-\infty, \infty)\) using the mapping

\[ h(u) = XScale \cdot \tan(\pi(u - 1/2)) \]

**Constructor**

ZerosFunction  
public ZerosFunction()

**Description**  
Creates an instance of the solver.
Methods

**ComputeZeros**

```csharp
virtual public double[] ComputeZeros(Imsl.Math.ZerosFunction.IFunction objectF)
```

**Description**

Returns the zeros of a univariate function.

**Parameter**

- `objectF` – Contains the function for which the zeros will be found

**Returns**

A `double` array containing the zeros of the univariate function.

---

**SetBounds**

```csharp
virtual public void SetBounds(double lowerBound, double upperBound)
```

**Description**

Sets the closed interval in which to search for the roots.

**Parameters**

- `lowerBound` – A `double` containing the lower interval bound. `lowerBound` cannot be greater than or equal to `upperBound`.
- `upperBound` – A `double` containing the upper interval bound.

**Remarks**

The function must be defined for all values in this interval.

Default: The search for the roots is not bounded.

---

**SetGuess**

```csharp
virtual public void SetGuess(double[] guess)
```

**Description**

Sets the initial guess for the zeros.

**Parameter**

- `guess` – A `double` array containing the initial guesses for the number of zeros to be found. If a bound on the zeros is also given, the guesses must satisfy the bound condition.

---

**Example: Zeros of a Univariate Function**

In this example 3 zeros of the `sin` function are found.

```csharp
using System;
using Imsl.Math;

public class ZerosFunctionEx1 : ZerosFunction.IFunction
```
```csharp
public virtual double F(double x)
{
    return Math.Sin(x);
}

public static void Main(String[] args)
{
    ZerosFunction.IFunction fcn = new ZerosFunctionEx1();

    ZerosFunction zf = new ZerosFunction();
    double[] guess = new double[]{5, 18, -6};
    zf.SetGuess(guess);
    double[] zeros = zf.ComputeZeros(fcn);
    for (int k = 0; k < zeros.Length; k++)
    {
        Console.Out.WriteLine(zeros[k] + " = " + (zeros[k] / Math.PI) + " pi");
    }
}
```

**Output**
-2.953824405026E-22 = -9.40231510170729E-23 pi
3.14159265358979 = 1 pi
6.28318530717959 = 2 pi

---

### ZerosFunction.IFunction Interface

**public interface Imsl.Math.ZerosFunction.IFunction**

Public interface for the user supplied function to ZerosFunction. The user supplied function, \( f(x) \), must return valid results for all values in the specified interval. If no interval is given, the user-supplied function must return valid results for all real numbers.

**Method**

**F**

abstract public double F(double x)

**Description**

Returns the value of the function at the given point.
Parameter

- A double specifying the point at which the function is to be evaluated.

Returns

A double containing the value of the function at x.

---

**ZeroSystem Class**

```java
public class Imsl.Math.ZeroSystem
```

Solves a system of n nonlinear equations f(x) = 0 using a modified Powell hybrid algorithm.

ZeroSystem is based on the MINPACK subroutine HYBRD1, which uses a modification of M.J.D. Powell’s hybrid algorithm. This algorithm is a variation of Newton’s method, which uses a finite-difference approximation to the Jacobian and takes precautions to avoid large step sizes or increasing residuals. For further description, see More et al. (1980).

A finite-difference method is used to estimate the Jacobian. Whenever the exact Jacobian can be easily provided, `f` should implement `ZeroSystem.IJacobian`.

Note that one can use logging to generate intermediate output for the solver. Accumulated levels of detail correspond to `Config`, `Fine`, `Finer`, and `Finest` logging levels with `Config` yielding the smallest amount of information and `Finest` yielding the most. The levels of output yield the following:

<table>
<thead>
<tr>
<th>Level</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>Iteration increments are printed.</td>
</tr>
<tr>
<td>Fine</td>
<td>Prints convergence tests.</td>
</tr>
<tr>
<td>Finer</td>
<td>Intermediate solution values are provided.</td>
</tr>
<tr>
<td>Finest</td>
<td>Tracks progress through internal methods.</td>
</tr>
</tbody>
</table>

**Properties**

**Logger**

```java
public Imsl.Logger Logger {get; set; }
```

**Description**

The Logger associated with this object.

**Property Value**

The Logger associated with this object.
Remarks
The Logger object enables logging.
Default: Logger = null

MaximumIterations
public int MaximumIterations {get; set; }

Description
The maximum number of iterations allowed.

Property Value
An int specifying the maximum number of iterations allowed.

Remarks
The default value is 200.

Exception
System.ArgumentException is thrown if MaxIterations is less than or equal to zero

RelativeError
public double RelativeError {get; set; }

Description
The relative error tolerance.

Property Value
A double specifying the relative error tolerance.

Remarks
The root is accepted if the relative error between two successive approximations to this root is within errorRelative. The default is the square root of the precision, about 1.0e-08.

Exception
System.ArgumentException is thrown if RelativeError is less than 0 or greater than 1

Constructor

ZeroSystem
public ZeroSystem(int n)

Description
Creates an object to find the zeros of a system of n equations.

Parameter
n – The number of equations that the solver handles.
Methods

**SetGuess**

public void SetGuess(double[] guess)

**Description**
Sets initial guess for the solution.

**Parameter**
- guess – A double array containing the initial guess.

**Solve**

public double[] Solve(Imsl.Math.ZeroSystem.IFunction f)

**Description**
Solve a system of nonlinear equations using the Levenberg-Marquardt algorithm.

**Parameter**
- f – Defines a ZeroSystem.IFunction whose zero is to be found. If f implements a ZeroSystem.IJacobian then its Jacobian is used. Otherwise finite difference is used.

**Returns**
A double array containing the solution.

**Exceptions**
- Imsl.Math.TooManyIterationsException is thrown if the maximum number of iterations is exceeded
- Imsl.Math.ToleranceTooSmallException is thrown if the error tolerance is too small
- Imsl.Math.DidNotConvergeException is thrown if the algorithm does not converge

See Also: Imsl.Math.ZeroSystem.IJacobian (p. 333)

**Example 1: Solve a System of Nonlinear Equations**
A system of nonlinear equations is solved.

```csharp
using System;
using Imsl.Math;

public class ZeroSystemEx1 : ZeroSystem.IFunction
{
    public void F(double[] x, double[] f)
    {
    }
}
```
public static void Main(String[] args)
{
    ZeroSystem zf = new ZeroSystem(3);
    zf.SetGuess(new double[]{4.0, 4.0, 4.0});
    new PrintMatrix("zeros").Print(zf.Solve(new ZeroSystemEx1()));
}

Output

zeros
0
0.99999999995498
1 2.00000000000656
2 2.99999999999468

Example 2: Solve a System of Nonlinear Equations with Logging

A system of nonlinear equations is solved with reduced accuracy and logging enabled.

using System;
using Imsl.Math;

public class ZeroSystemEx2 : ZeroSystem.IFunction
{
    public void F(double[] x, double[] f)
    {
        f[0] = 0.5 * x[0] + x[1] + 0.5 * x[2] - x[5] / x[6];
    }

    public static void Main(String[] args)
    {
        ZeroSystem zf = new ZeroSystem(7);
        zf.RelativeError = 0.01; // reduced accuracy
        double[] guess = {0.5, 0.0, 0.0, 0.5, 0.0, 0.5, 2.0};
        zf.SetGuess(guess);

        // enable logging to the console
        zf.Logger = new Imsl.Logger();
        zf.Logger.LogLevel = Imsl.Logger.Level.Finer;
    }
}
new PrintMatrix("zeros").Print(zf.Solve(new ZeroSystemEx2()));
}
}

Output
ZeroSystem: Iteration 1
ZeroSystem: Current solution:
0.5
ZeroSystem: Convergence if 0.214305934943747 <= 0.0217944947177034
ZeroSystem: Convergence if 881.854579848628 == 0.0
ZeroSystem: Convergence if 0.428611869887493 <= 0.0237140589837329
ZeroSystem: Convergence if 62.2023634352405 == 0.0
ZeroSystem: Iteration 2
ZeroSystem: Current solution:
0.451941633824387
ZeroSystem: Convergence if 0.214305934943747 <= 0.0237140589837329
ZeroSystem: Convergence if 62.2023634352405 == 0.0
ZeroSystem: Convergence if 0.214305934943747 <= 0.0256689283849008
ZeroSystem: Convergence if 48.6311502628328 == 0.0
ZeroSystem: Iteration 3
ZeroSystem: Current solution:
0.410355683664779
ZeroSystem: Convergence if 0.107152967471873 <= 0.0256689283849008
ZeroSystem: Convergence if 48.6311502628328 == 0.0
ZeroSystem: Convergence if 0.214305934943747 <= 0.026655240629938
ZeroSystem: Convergence if 24.594369376895 == 0.0
ZeroSystem: Iteration 4
ZeroSystem: Current solution:
0.389763013577616
ZeroSystem: Convergence if 0.107152967471873 <= 0.026655240629938
ZeroSystem: Convergence if 24.594369376895 == 0.0
ZeroSystem: Convergence if 0.214305934943747 <= 0.0276551655584895
ZeroSystem: Convergence if 11.233356734159 == 0.0
ZeroSystem: Iteration 5
ZeroSystem: Current solution:
0.371417874014809
ZeroSystem: Convergence if 0.107152967471873 <= 0.0276551655584895
ZeroSystem: Convergence if 11.233356734159 == 0.0
ZeroSystem: Convergence if 0.214305934943747 <= 0.0286672864029196
ZeroSystem: Convergence if 4.27372353401102 == 0.0
ZeroSystem: Iteration 6
ZeroSystem: Current solution:
0.355229191830464
ZeroSystem: Convergence if 0.107152967471873 <= 0.0286672864029196
ZeroSystem: Convergence if 4.27372353401102 == 0.0
ZeroSystem: Convergence if 0.053576483759367 <= 0.0286672864029196
ZeroSystem: Convergence if 4.27372353401102 == 0.0
ZeroSystem: Iteration 6
ZeroSystem: Current solution:
0.355229191830464
ZeroSystem: Convergence if 0.107152967471873 <= 0.0291782082184281
ZeroSystem: Convergence if 1.33974481695402 == 0.0
ZeroSystem: Iteration 7
ZeroSystem: Current solution:
ZeroSystem.IFunction Interface

public interface Imsl.Math.ZeroSystem.IFunction

Public interface for user supplied function to ZeroSystem object.

Method

F
abstract public void F(double[] x, double[] fvalue)

Description
On return, fvalue contains the function value at the given point.

Parameters

x – A double array which contains the point at which the function is to be evaluated. The contents
of this array must not be altered by this function.

fvalue – A double array which, on return, contains the value of the function at x.
ZeroSystem.IJacobian Interface

public interface Imsl.Math.ZeroSystem.IJacobian :
Imsl.Math.ZeroSystem.IFunction

Public interface for user supplied function to ZeroSystem object.

Method

Jacobian
abstract public void Jacobian(double[] x, double[,] jac)

Description
On return, jac contains the value of the Jacobian at the given point.

Parameters

  x – A double array which contains the point at which the Jacobian is to be evaluated. The contents of this array must not be altered by this function.
  jac – A double matrix which, on return, contains the value of the Jacobian at x. The value of jac[i,j] is the derivative of f[i] with respect to x[j].
Chapter 8: Optimization

Types

class MinUncon ................................................................. 338
interface MinUncon(IFunction ........................................... 343
interface MinUncon(IDerivative .......................................... 344
class MinUnconMultiVar .................................................. 344
interface MinUnconMultiVar(IFunction ............................... 352
interface MinUnconMultiVar(IGradient ............................... 352
class NonlinLeastSquares ............................................... 353
interface NonlinLeastSquares(IFunction ......................... 362
interface NonlinLeastSquares(IJacobian ......................... 362
class DenseLP ............................................................. 363
class MPSReader ............................................................ 371
class MPSReader.Element ................................................ 383
class MPSReader.Row ..................................................... 384
class QuadraticProgramming .......................................... 385
class MinConGenLin ....................................................... 391
interface MinConGenLin(IFunction ............................... 399
interface MinConGenLin(IGradient ............................... 399
class BoundedLeastSquares ............................................ 400
interface BoundedLeastSquares(IFunction ....................... 408
interface BoundedLeastSquares(IJacobian ....................... 409
class BoundedVariableLeastSquares .................................. 409
class NonNegativeLeastSquares ....................................... 413
class MinConNLP .......................................................... 417
interface MinConNLP(IFunction ...................................... 429
interface MinConNLP(IGradient ...................................... 430
class NumericalDerivatives ............................................ 431
interface NumericalDerivatives(IFunction ..................... 449
interface NumericalDerivatives(IJacobian ..................... 450
enumeration NumericalDerivatives.DifferencingMethod ....... 451
Usage Notes

Unconstrained Minimization

The unconstrained minimization problem can be stated as follows:

$$\min_{x \in \mathbb{R}^n} f(x)$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is continuous and has derivatives of all orders required by the algorithms. The functions for unconstrained minimization are grouped into three categories: univariate functions, multivariate functions, and nonlinear least-squares functions.

For the univariate functions, it is assumed that the function is unimodal within the specified interval. For discussion on unimodality, see Brent (1973).

The class `MinUnconMultiVar` finds the minimum of a multivariate function using a quasi-Newton method. The default is to use a finite-difference approximation of the gradient of $f(x)$. Here, the gradient is defined to be the vector

$$\nabla f(x) = \left[ \frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n} \right]$$

However, when the exact gradient can be easily provided, the gradient should be provided by implementing the interface `MinUnconMultiVar.Gradiant`. The `NumericalDerivatives` class can also be used in computing the gradient for `MinUnconMultiVar` as well as other classes. For an example, see `NumericalDerivatives` Example 6.

The nonlinear least-squares function uses a modified Levenberg-Marquardt algorithm. The most common application of the function is the nonlinear data-fitting problem where the user is trying to fit the data with a nonlinear model.

These functions are designed to find only a local minimum point. However, a function may have many local minima. Try different initial points and intervals to obtain a better local solution.

Linearly Constrained Minimization

The linearly constrained minimization problem can be stated as follows:

$$\min_{x \in \mathbb{R}^n} f(x)$$

subject to $A_1 x = b_1$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $A_1$ is a coefficient matrix, and $b_1$ is a vector. If $f(x)$ is linear, then the problem is a linear programming problem. If $f(x)$ is quadratic, the problem is a quadratic programming problem.

The class `DenseLP` can be used to solve small- to medium-sized linear programming problems. No sparsity is assumed since the coefficients are stored in full matrix form.

The class `QuadraticProgramming` is designed to solve convex quadratic programming problems using a dual quadratic programming algorithm. If the given Hessian is not positive definite, then
QuadraticProgramming modifies it to be positive definite. In this case, output should be interpreted
with care because the problem has been changed slightly. Here, the Hessian of \( f(x) \) is defined to be the \( n \times n \) matrix

\[
\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2}{\partial x_i \partial x_j} f(x) \end{bmatrix}
\]

**Nonlinearly Constrained Minimization**

The nonlinearly constrained minimization problem can be stated as follows:

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad f(x) \\
\text{subjectto} & \quad g_i(x) = 0 \text{ for } i = 1, 2, \ldots, m_1 \\
& \quad g_i(x) \geq 0 \text{ for } i = m_1 + 1, \ldots, m
\end{align*}
\]

where \( f : \mathbb{R}^n \to \mathbb{R} \) and \( g_i : \mathbb{R}^n \to \mathbb{R} \), for \( i = 1, 2, \ldots, m \).

The class MinConNLP uses a sequential equality constrained quadratic programming algorithm to solve
this problem. A more complete discussion of this algorithm can be found in the documentation.

**Return Values from User-Supplied Functions**

All values returned by user-supplied functions must be valid real numbers. It is the user’s responsibility
to check that the values returned by a user-supplied function do not contain NaN, infinity, or negative
infinity values.

**Example: Minimum of a smooth function**

The minimum of \( e^x - 5x \) is found using function evaluations only.

```csharp
using System;
using Imsl.Math;

public class OptimizationIntroEx1 : MinUncon.IFunction
{
    public double F(double x)
    {
        double y = Math.Exp(x) - 5.0 * x;
        if (!Double.IsNaN(y))
        {
            return y;
        }
        else
        {
            return 0.0;
        }
    }

    public static void Main(String[] args)
    {
        MinUncon zf = new MinUncon();
```
MinUnconClass

public class Imsl.Math.MinUncon

Finds the minimum point for a smooth univariate function using function and optionally first derivative evaluations.

MinUncon uses two separate algorithms to compute the minimum depending on what the user supplies as the function f.

If f defines the function whose minimum is to be found MinUncon uses a safeguarded quadratic interpolation method to find a minimum point of a univariate function. Both the code and the underlying algorithm are based on the routine ZXLSF written by M.J.D. Powell at the University of Cambridge.

MinUncon finds the least value of a univariate function, f, where f is a MinUncon.IFunction. Optional data include an initial estimate of the solution, and a positive number specified by the Bound property.

Let x₀ = Guess where Guess is specified by the Guess property and b = Bound, then x is restricted to the interval [x₀ - b, x₀ + b]. Usually, the algorithm begins the search by moving from x₀ to x = x₀ + s, where s = Step. Step is set by the Step property. If Step is not called then Step is set to 0.1. Step may be positive or negative. The first two function evaluations indicate the direction to the minimum point, and the search strides out along this direction until a bracket on a minimum point is found or until x reaches one of the bounds x₀ ± b. During this stage, the step length increases by a factor of between two and nine per function evaluation; the factor depends on the position of the minimum point that is predicted by quadratic interpolation of the three most recent function values.

When an interval containing a solution has been found, we will have three points, x₁, x₂, and x₃, with x₁ < x₂ < x₃ and f(x₂) ≤ f(x₁) and f(x₂) ≤ f(x₃). There are three main ingredients in the technique for choosing the new x from these three points. They are (i) the estimate of the minimum point that is given by quadratic interpolation of the three function values, (ii) a tolerance parameter ε, that depends on the closeness of f to a quadratic, and (iii) whether x₂ is near the center of the range between x₁ and x₃ or is relatively close to an end of this range. In outline, the new value of x is as near as possible to the predicted minimum point, subject to being at least ε from x₂, and subject to being in the longer interval between x₁ and x₂ or x₂ and x₃ when x₂ is particularly close to x₁ or x₃. There is some elaboration, however, when the distance between these points is close to the required accuracy; when the distance is close to the machine precision; or when ε is relatively large.

The algorithm is intended to provide fast convergence when f has a positive and continuous second derivative at the minimum and to avoid gross inefficiencies in pathological cases, such as
\( f(x) = x + 1.001|x| \)

The algorithm can make \( \epsilon \) large automatically in the pathological cases. In this case, it is usual for a new value of \( x \) to be at the midpoint of the longer interval that is adjacent to the least calculated function value. The midpoint strategy is used frequently when changes to \( f \) are dominated by computer rounding errors, which will almost certainly happen if the user requests an accuracy that is less than the square root of the machine precision. In such cases, the routine claims to have achieved the required accuracy if it knows that there is a local minimum point within distance \( \delta \) of \( x \), where \( \delta = x_{acc} \), specified by the Accuracy property even though the rounding errors in \( f \) may cause the existence of other local minimum points nearby. This difficulty is inevitable in minimization routines that use only function values, so high precision arithmetic is recommended.

If \( f \) is a MinUncon.IDerivative then MinUncon uses a descent method with either the secant method or cubic interpolation to find a minimum point of a univariate function. It starts with an initial guess and two endpoints. If any of the three points is a local minimum point and has least function value, the routine terminates with a solution. Otherwise, the point with least function value will be used as the starting point.

From the starting point, say \( x_c \), the function value \( f_c = f(x_c) \), the derivative value \( g_c = g(x_c) \), and a new point \( x_n \) defined by \( x_n = x_c - g_c \) are computed. The function \( f_n = f(x_n) \), and the derivative \( g_n = g(x_n) \) are then evaluated. If either \( f_n \geq f_c \) or \( g_n \) has the opposite sign of \( g_c \), then there exists a minimum point between \( x_c \) and \( x_n \); and an initial interval is obtained. Otherwise, since \( x_c \) is kept as the point that has lowest function value, an interchange between \( x_n \) and \( x_c \) is performed. The secant method is then used to get a new point

\[
x_s = x_c - g_c \left( \frac{g_n - g_c}{x_n - x_c} \right)
\]

Let \( x_n \leftarrow x_s \) and repeat this process until an interval containing a minimum is found or one of the convergence criteria is satisfied. The convergence criteria are as follows:

Criterion 1:

\[
|x_c - x_n| \leq \epsilon_c
\]

Criterion 2:

\[
|g_c| \leq \epsilon_g
\]

where \( \epsilon_c = \max \{ 1.0, |x_c| \} \; \epsilon \), \( \epsilon \) is a relative error tolerance and \( \epsilon_c \) is a gradient tolerance.

When convergence is not achieved, a cubic interpolation is performed to obtain a new point. The function and derivative are then evaluated at that point; and accordingly, a smaller interval that contains a
minimum point is chosen. A safeguarded method is used to ensure that the interval reduces by at least a fraction of the previous interval. Another cubic interpolation is then performed, and this procedure is repeated until one of the stopping criteria is met.

**Properties**

**Accuracy**

```csharp
public double Accuracy { get; set; }
```

**Description**
The required absolute accuracy in the final value returned by the `ComputeMin` method.

**Property Value**
A double scalar value specifying the required absolute accuracy in the final value returned by the `ComputeMin` method.

**Remarks**
By default, the required accuracy is set to 1.0e-8.

**Bound**

```csharp
public double Bound { get; set; }
```

**Description**
The amount by which X may be changed from its initial value, Guess.

**Property Value**
A double scalar value specifying the amount by which X may be changed from its initial value. In other words, X is restricted to the interval [Guess-Bound, Guess+Bound].

**Remarks**
By default, Bound is set to 100.

**DerivTolerance**

```csharp
public double DerivTolerance { get; set; }
```

**Description**
The derivative tolerance used by member method `ComputeMin` to decide if the current point is a local minimum.

**Property Value**
A double scalar value specifying the derivative tolerance used by member method `ComputeMin`.

**Remarks**
This is the second stopping criterion. x is returned as a solution when G(x) is less than or equal to DerivTolerance. DerivTolerance should be nonnegative, otherwise zero will be used. By default, DerivTolerance is set to 1.0e-8.
**Guess**

```csharp
public double Guess {get; set; }
```

**Description**
The initial guess of the minimum point of the input function.

**Property Value**
A double scalar value specifying the initial guess of the minimum point of the input function.

**Remarks**
By default, an initial guess of 0.0 is used.

---

**Step**

```csharp
public double Step {get; set; }
```

**Description**
The stepsize to use when changing $x$.

**Property Value**
A double scalar value specifying the order of magnitude estimate of the required change in $x$ when stepping towards the minimum.

**Remarks**
By default, Step is set to 0.1.

---

**Constructor**

**MinUncon**

```csharp
public MinUncon()
```

**Description**
Unconstrained minimum constructor for a smooth function of a single variable of type double.

---

**Method**

**ComputeMin**

```csharp
public double ComputeMin(Imsl.Math.MinUncon.IFunction f)
```

**Description**
Return the minimum of a smooth function of a single variable of type double using function values only or using function values and derivatives.
Parameter

$f$ – The MinUncon.IFunction whose minimum is to be found. An attempt to find the minimum is made using function values only.

Returns

A double scalar value containing the minimum of the input function.

Example 1: Minimum of a smooth function

The minimum of $e^x - 5x$ is found using function evaluations only.

```csharp
using System;
using Imsl.Math;

public class MinUnconEx1 : MinUncon.IFunction
{
    public double F(double x)
    {
        return Math.Exp(x) - 5.0 * x;
    }

    public static void Main(String[] args)
    {
        MinUncon zf = new MinUncon();
        zf.Guess = 0.0;
        zf.Accuracy = 0.001;
        MinUncon.IFunction fcn = new MinUnconEx1();
        Console.Out.WriteLine("Minimum is " + zf.ComputeMin(fcn));
    }
}
```

Output

Minimum is 1.60941759992002

Example 2: Minimum of a smooth function

The minimum of $e^x - 5x$ is found using function evaluations and first derivative evaluations.

```csharp
using System;
using Imsl.Math;

public class MinUnconEx2 : MinUncon.IDerivative
{
    public double F(double x)
    {
        return Math.Exp(x) - 5.0 * x;
    }

    public double Derivative(double x)
    {
        return Math.Exp(x) - 5.0;
    }
}
```
```csharp
    { return Math.Exp(x) - 5.0;
    }

    public static void Main(String[] args)
    {
        MinUncon zf = new MinUncon();
        zf.Guess = 0.0;
        zf.Accuracy = .001;
        double x = zf.ComputeMin(new MinUnconEx2());
        Console.Out.WriteLine("x = " + x);
    }
}

Output
x = 1.61001131622703

MinUncon.IFunction Interface

public interface Imsl.Math.MinUncon.IFunction
Interface for the user supplied function for the smooth function of a single variable to be minimized.

Method

F
abstract public double F(double x)

Description
Smooth function of a single variable to be minimized.

Parameter
x – A double, the point at which the function is to be evaluated.

Returns
A double, the value of the function at x.
MinUncon.IDerivative Interface

Interface for the smooth function of a single variable to be minimized and its derivative.

Method

Derivative
abstract public double Derivative(double x)

Description
Derivative of the smooth function of a single variable to be minimized.

Parameter
x – A double, the point at which the derivative of the function is to be evaluated.

Returns
A double, the value of the derivative of the function at x.

MinUnconMultiVar Class

public class Imsl.Math.MinUnconMultiVar
Minimizes a multivariate function using a quasi-Newton method.

Class MinUnconMultiVar uses a quasi-Newton method to find the minimum of a function \( f(x) \) of \( n \) variables. The problem is stated as follows:

\[
\min_{x \in \mathbb{R}^n} f(x)
\]

Given a starting point \( x_c \), the search direction is computed according to the formula

\[
d = -B^{-1}g_c
\]

where \( B \) is a positive definite approximation of the Hessian, and \( g_c \) is the gradient evaluated at \( x_c \). A line search is then used to find a new point.
\[ x_n = x_c + \lambda d, \lambda > 0 \]

such that

\[ f(x_n) \leq f(x_c) + \alpha g^T d, \alpha \in (0, 0.5) \]

Finally, the optimality condition \(||g(x)|| \leq \varepsilon||\ where \ \varepsilon is a gradient tolerance. When optimality is not achieved, \( B \) is updated according to the BFGS formula

\[
B \leftarrow B - \frac{Bss^T B}{s^T B s} + \frac{yy^T}{y^T s}
\]

where \( s = x_n - x_c \) and \( y = g_n - g_c \). Another search direction is then computed to begin the next iteration. For more details, see Dennis and Schnabel (1983, Appendix A).

In this implementation, the first stopping criterion for MinUnconMultivar occurs when the norm of the gradient is less than the given gradient tolerance property, GradientTolerance. The second stopping criterion for MinUnconMultivar occurs when the scaled distance between the last two steps is less than the step tolerance property, StepTolerance.

Since by default, a finite-difference method is used to estimate the gradient. An inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. Supply the gradient for a more accurate gradient evaluation (MinConMultiVar.IGradient).

**Properties**

**Digits**

```csharp
public double Digits {get; set; }
```

**Description**

The number of good digits in the function.

**Property Value**

A double scalar value specifying the number of good digits in the user supplied function.

**Remarks**

By default, Digits is set to 15.75.

**Exception**

System.ArgumentException is thrown if Digits is less than or equal to 0.
**ErrorStatus**

public int ErrorStatus {get; }

**Description**
The non-fatal error status.

**Property Value**
An int specifying the non-fatal error status:

<table>
<thead>
<tr>
<th>ErrorStatus</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The last global step failed to locate a lower point than the current ( x ) value. The current ( x ) may be an approximate local minimizer and no more accuracy is possible or the step tolerance may be too large.</td>
</tr>
<tr>
<td>2</td>
<td>Relative function convergence; both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.</td>
</tr>
<tr>
<td>3</td>
<td>Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or the step tolerance is too big.</td>
</tr>
</tbody>
</table>

**Fsxle**

public double Fsxle {get; set; }

**Description**
The function scaling value for scaling the gradient.

**Property Value**
A double scalar specifying the function scaling value for scaling the gradient.

**Remarks**
By default, the value of this scalar is set to 1.0.

**Exception**

System.ArgumentException is thrown if Fsxle is less than or equal to 0

**GradientTolerance**

public double GradientTolerance {get; set; }

**Description**
The gradient tolerance used to compute the gradient.

**Property Value**
A double specifying the gradient tolerance used to compute the gradient.

**Remarks**
By default, the cube root of machine precision squared is used to compute the gradient.
Exception
    System.ArgumentException is thrown if GradientTolerance is less than or equal to 0

**Ihess**
public int Ihess {get; set; }

**Description**
The Hessian initialization parameter.

**Property Value**
An int scalar value specifying the Hessian initialization parameter.

**Remarks**
By default, Ihess is set to 0.0 and the Hessian is initialized to the identity matrix. If this member function is called and Ihess is set to anything other than 0.0, the Hessian is initialized to the diagonal matrix containing
\[
\text{\text{max(abs(f(xguess)),fscale)}*xscale*xscale}
\]
where xguess is the initial guess of the computed solution and xscale is the scaling vector for the variables.

**Iterations**
public int Iterations {get; }

**Description**
The number of iterations used to compute a minimum.

**Property Value**
An int specifying the number of iterations used to compute the minimum.

**MaximumStepsize**
public double MaximumStepsize {get; set; }

**Description**
The maximum allowable stepsize to use.

**Property Value**
A nonnegative double value specifying the maximum allowable stepsize.

**Remarks**
By default, maximum stepsize is set to a value based on a scaled Guess.

**Exception**
    System.ArgumentException is thrown if MaximumStepsize is less than or equal to 0

**MaxlIterations**
public int MaxlIterations {get; set; }

**Description**
The maximum number of iterations allowed.
**MaxIterations**
An int specifying the maximum number of iterations allowed.

**Remarks**
By default, the maximum number of iterations is set to 100.

**Exception**
System.ArgumentException is thrown if MaxIterations is less than or equal to 0

---

**NumberOfProcessors**
public int NumberOfProcessors {get; set; }

**Description**
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**
An int indicating the maximum possible number of processors to use.

**Remarks**
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

---

**Parallel**
public bool Parallel {get; set; }

**Description**
Enable or disable performing MinUnconMultiVar.IFunction.F in parallel.

**Property Value**
A bool indicating whether or not the MinUnconMultiVar.IFunction.F calculations are to be performed in parallel.

**Remarks**
By default, Parallel = true. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

---

**StepTolerance**
public double StepTolerance {get; set; }

**Description**
The scaled step tolerance to use when changing x.

**Property Value**
A double scalar value specifying the scaled step tolerance.
Remarks
The i-th component of the scaled step between two points \( x \) and \( y \) is computed as
\[
\frac{\text{abs}(x(i)-y(i))}{\max(\text{abs}(x(i)),1/xscale(i))}
\]
where \( xscale \) is the scaling vector for the variables.
By default, the scaled step tolerance is set to 3.66685e-11.

Exception
System.ArgumentException is thrown if StepTolerance is less than or equal to 0

Constructor

**MinUnconMultiVar**

```java
public MinUnconMultiVar(int n)
```

Description
Unconstrained minimum constructor for a function of \( n \) variables of type double.

Parameter
- \( n \) – An int scalar value which defines the number of variables of the function whose minimum is to be found.

Methods

**ComputeMin**

```java
public double[] ComputeMin(Imsl.Math.MinUnconMultiVar.IFunction f)
```

Description
Return the minimum point of a function of \( n \) variables of type double using a finite-difference gradient or using a user-supplied gradient.

Parameter
- \( f \) – The MinUnconMultiVar.IFunction whose minimum is to be found.

Returns
A double array containing the point at which the minimum of the input function occurs.

Remarks
\( f \) can be used to supply a gradient of the function. If \( f \) implements IGradient then the user-supplied gradient is used. Otherwise, an attempt to find the minimum is made using a finite-difference gradient.
Exceptions

- `Imsl.Math.FalseConvergenceException` is thrown if the iterates appear to be converging to a noncritical point
- `Imsl.Math.MaxIterationsException` is thrown if the maximum number of iterations is exceeded
- `Imsl.Math.UnboundedBelowException` is thrown if five consecutive steps of the maximum allowable stepsize have been taken, either the function is unbounded below, or has a finite asymptote in some direction or the maximum allowable step size is too small

**SetGuess**

```csharp
public void SetGuess(double[] guess)
```

**Description**
Sets the initial guess of the minimum point of the input function.

**Parameter**
- `guess` – A `double` array specifying the initial guess of the minimum point of the input function.

**Remarks**
By default, the elements of this array are set to 0.0.

**SetXscale**

```csharp
public void SetXscale(double[] xscale)
```

**Description**
Sets the diagonal scaling matrix for the variables.

**Parameter**
- `xscale` – A `double` array specifying the diagonal scaling matrix for the variables.

**Remarks**
By default, the elements of this array are set to 1.0.

**Exception**
- `System.ArgumentException` is thrown if any of the elements of `Xscale` is less than or equal to 0

**Example 1: Minimum of a multivariate function**
The minimum of \(100(x_2 - x_1^2)^2 + (1 - x_1)^2\) is found using function evaluations only.

```csharp
using System;
using Imsl.Math;

public class MinUnconMultiVarEx1 : MinUnconMultiVar.IFunction
{
    public double F(double[] x)
    {
```
Example 2: Minimum of a multivariate function

The minimum of \(100(x_2 - x_1^2)^2 + (1 - x_1)^2\) is found using function evaluations and a user supplied gradient.

```csharp
using System;
using Imsl.Math;

public class MinUnconMultiVarEx2 : MinUnconMultiVar.IGradient
{
    public double F(double[] x)
    {
        return 100.0 * ((x[1] - x[0] * x[0]) * (x[1] - x[0] * x[0])) +
               (1.0 - x[0]) * (1.0 - x[0]);
    }

    public void Gradient(double[] x, double[] gp)
    {
        gp[0] = -400.0 * (x[1] - x[0] * x[0]) * x[0] - 2.0 * (1.0 - x[0]);
        gp[1] = 200.0 * (x[1] - x[0] * x[0]);
    }
}
```

```csharp
public static void Main(String[] args)
{
    MinUnconMultiVar solver = new MinUnconMultiVar(2);
    solver.SetGuess(new double[]{ -1.2, 1.0 });
    double[] x = solver.ComputeMin(new MinUnconMultiVarEx2());
    Console.Out.WriteLine
        ("Minimum point is (" + x[0] + ", " + x[1] + ")");
}
```

Output

Minimum point is (0.99999996726513, 0.999999933045216)

Optimization
Output
Minimum point is (0.999999966882301, 0.999999932254245)

MinUnconMultiVar.IFunction Interface

public interface Imsl.Math.MinUnconMultiVar.IFunction
Interface for the user supplied multivariate function to be minimized.

Method

F
abstract public double F(double[] x)

Description
Multivariate function to be minimized.

Parameter
x – A double array, the point at which the function is to be evaluated.

Returns
A double, the value of the function at x.

MinUnconMultiVar.IGradient Interface

public interface Imsl.Math.MinUnconMultiVar.IGradient :
Imsl.Math.MinUnconMultiVar.IFunction
Interface for the user supplied multivariate function to be minimized and its gradient.

Method

Gradient
abstract public void Gradient(double[] x, double[] gvalue)
Description
On return, gvalue contains the value of the gradient, of the function, at x.

Parameters
x – A double array, the point at which the gradient of the function is to be evaluated.
gvalue – A double array which, on return, contains the value of the gradient, of the function, at x.

NonlinLeastSquares Class

public class Imsl.Math.NonlinLeastSquares
Solves a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm.

NonlinLeastSquares is based on the MINPACK routine LMDIF by More et al. (1980). It uses a modified Levenberg-Marquardt method to solve nonlinear least squares problems. The problem is stated as follows:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

where $m \geq n$, $F : \mathbb{R}^n \to \mathbb{R}^m$, and $f_i(x)$ is the i-th component function of $F(x)$. From a current point, the algorithm uses the trust region approach:

$$\min_{x_n \in \mathbb{R}^n} \| F(x_c) + J(x_c)(x_n - x_c) \|_2$$

subject to

$$\|x_n - x_c\|_2 \leq \delta_c$$

to get a new point $x_n$, which is computed as

$$x_n = x_c - \left( J(x_c)^T J(x_c) + \mu_c I \right)^{-1} J(x_c)^T F(x_c)$$

where $\mu_c = 0$ if $\delta_c \geq \| J(x_c)^T J(x_c) \|^{-1} J(x_c)^T F(x_c) \|_2$ and $\mu_c > 0$ otherwise. $F(x_c)$ and $J(x_c)$ are the function values and the Jacobian evaluated at the current point $x_c$. This procedure is repeated until the stopping criteria are satisfied. The first stopping criteria occurs when the norm of the function is less than the property AbsoluteTolerance. The second stopping criteria occurs when the norm of the
scaled gradient is less than the property GradientTolerance. The third stopping criteria occurs when the scaled distance between the last two steps is less than the property StepTolerance. For more details, see Levenberg (1944), Marquardt (1963), or Dennis and Schnabel (1983, Chapter 10).

A finite-difference method is used to estimate the Jacobian when the user supplied function, $f$, defines the least-squares problem. Whenever the exact Jacobian can be easily provided, $f$ should implement NonlinLeastSquares.Jacobian.

**Properties**

**AbsoluteTolerance**

```csharp
public double AbsoluteTolerance {get; set; }
```

**Description**

The absolute function tolerance.

**Property Value**

A double scalar value specifying the absolute function tolerance.

**Remarks**

When the norm of the function is less than AbsoluteTolerance the procedure stops.

By default, AbsoluteTolerance = 1.0e-32.

**Exception**

System.ArgumentException is thrown if AbsoluteTolerance is less than or equal to 0

**Digits**

```csharp
virtual public int Digits {get; set; }
```

**Description**

The number of good digits in the function.

**Property Value**

An int specifying the number of good digits in the user supplied function which defines the least-squares problem.

**Remarks**

By default, the number of good digits is set to 7.

**Exception**

System.ArgumentException is thrown if Digits is less than or equal to 0

**ErrorStatus**

```csharp
public int ErrorStatus {get; }
```

**Description**

Get information about the performance of NonlinLeastSquares.
Property Value
An int specifying information about convergence.

Remarks

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All convergence tests were met.</td>
</tr>
<tr>
<td>1</td>
<td>Scaled step tolerance was satisfied. The current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or StepTolerance is too big.</td>
</tr>
<tr>
<td>2</td>
<td>Scaled actual and predicted reductions in the function are less than or equal to the relative function convergence tolerance RelativeTolerance.</td>
</tr>
<tr>
<td>3</td>
<td>Iterates appear to be converging to a noncritical point. Incorrect gradient information, a discontinuous function, or stopping tolerances being too tight may be the cause.</td>
</tr>
<tr>
<td>4</td>
<td>Five consecutive steps with the maximum stepsize have been taken. Either the function is unbounded below, or has a finite asymptote in some direction, or the maximum stepsize is too small.</td>
</tr>
</tbody>
</table>


FalseConvergenceTolerance
public double FalseConvergenceTolerance {get; set; }

Description
The false convergence tolerance.

Property Value
A double scalar value specifying the false convergence tolerance.

Remarks
By default, 100.0e-16 is used as the false convergence tolerance.

Exception
System.ArgumentException is thrown if FalseConvergenceTolerance is less than or equal to 0

GradientTolerance
public double GradientTolerance {get; set; }

Description
The scaled gradient tolerance.
Property Value
A double specifying the scaled gradient tolerance used in stopping criteria. The $i$-th component of the scaled gradient at $x$ is calculated as

$$|g_i| \cdot \max(|x_i|, \frac{1}{s})$$

$$\frac{1}{s} ||F(x)||_2^2$$

where $g = \nabla F(x)$, $s = xscale$, and

$$||F(x)||_2^2 = \sum_{i=1}^{m} f_i(x)^2$$

where $f = fscale$.

Remarks
When the norm of the function is less than GradientTolerance the procedure stops.

By default, GradientTolerance = $\sqrt{\varepsilon}$ where $\varepsilon$ is machine precision.

Exception
System.ArgumentException is thrown if GradientTolerance is less than or equal to 0

InitialTrustRegion
public double InitialTrustRegion {get; set; }

Description
The initial trust region radius.

Property Value
A double scalar value specifying the initial trust region radius.

Remarks
By default, InitialTrustRegion is set based on the initial scaled Cauchy step.

Exception
System.ArgumentException is thrown if InitialTrustRegion is less than or equal to 0

MaximumIterations
public int MaximumIterations {get; set; }

Description
The maximum number of iterations allowed.

Property Value
An int specifying the maximum number of iterations allowed.

Remarks
By default, the maximum number of iterations is set to 100.

Exception
System.ArgumentException is thrown if MaxIterations is less than or equal to 0

MaximumStepsize
public double MaximumStepsize {get; set; }

356 • NonlinLeastSquares  IMSL C# Numerical Library
**Description**
The maximum allowable stepsize to use.

**Property Value**
A nonnegative double value specifying the maximum allowable stepsize.

**Remarks**
By default, the maximum stepsize is set to a default value based on a scaled Guess.

**Exception**
System.ArgumentException is thrown if MaximumStepsize is less than or equal to 0

---

**NumberOfProcessors**
public int NumberOfProcessors {get; set; }

**Description**
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**
An int indicating the maximum possible number of processors to use.

**Remarks**
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

---

**Parallel**
public bool Parallel {get; set; }

**Description**
Enable or disable performing NonlinLeastSquares.IFunction.F in parallel.

**Property Value**
A bool indicating whether or not the NonlinLeastSquares.IFunction.F calculations are to be performed in parallel.

**Remarks**
By default, Parallel = true. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

---

**RelativeTolerance**
public double RelativeTolerance {get; set; }

**Description**
The relative function tolerance.

**Property Value**
A double scalar value specifying the relative function tolerance.
Remarks
By default, RelativeTolerance = 1.0e-20.

Exception
System.ArgumentException is thrown if RelativeTolerance is less than or equal to 0

StepTolerance
public double StepTolerance {get; set; }

Description
The scaled step tolerance.

Property Value
A double scalar value specifying the scaled step tolerance used in stopping criteria. The $i$-th component of the scaled step between two points $x$ and $y$ is computed as
\[
\frac{|x_i - y_i|}{\max(|x_i|, s_i)}
\]
where $s = xscale$

Remarks
When the scaled distance between the last two steps is less than StepTolerance the procedure stops.
By default, StepTolerance = $\sqrt[3]{\varepsilon}$ where $\varepsilon$ is machine precision.

Exception
System.ArgumentException is thrown if StepTolerance is less than or equal to 0

Constructor

NonlinLeastSquares
public NonlinLeastSquares(int m, int n)

Description
Creates an object to solve a nonlinear least squares problem.

Parameters
- m – The number of functions
- n – The number of variables. n must be less than or equal to m.

Methods

SetFscale
public void SetFscale(double[] fscale)
Description
Sets the diagonal scaling matrix for the functions.

Parameter
- `fscale` – A double array specifying the diagonal scaling matrix for the functions. The i-th component of `fscale` is a positive scalar specifying the reciprocal magnitude of the i-th component function of the problem.

Remarks
By default, the identity is used.

Exception
- `System.ArgumentException` is thrown if any of the elements of `fscale` is less than or equal to 0

SetGuess
public void SetGuess(double[] guess)

Description
Sets the initial guess of the minimum point of the input function.

Parameter
- `guess` – A double array specifying the initial guess of the minimum point of the input function.

Remarks
By default, an initial guess of 0.0 is used.

SetXscale
public void SetXscale(double[] xscale)

Description
Set the diagonal scaling matrix for the variables.

Parameter
- `xscale` – A double array specifying the diagonal scaling matrix for the variables. `xscale` is used in scaling the gradient and the distance between two points. See properties GradientTolerance and StepTolerance for more detail.

Remarks
By default, the identity is used.

Exception
- `System.ArgumentException` is thrown if any of the elements of `xscale` is less than or equal to 0

Solve
public double[] Solve(Imsl.Math.NonlinLeastSquares.IFunction f)
Description
Solve a nonlinear least-squares problem using a modified Levenberg-Marquardt algorithm and a Jacobian.

Parameter
- f – User supplied NonlinLeastSquares.IFunction that defines the least-squares problem. If f implements IJacobian then its Jacobian is used. Otherwise, a finite difference Jacobian is used.

Returns
A double array of length n containing the approximate solution.

Exception
Imsl.Math.TooManyIterationsException is thrown if the number of iterations exceeds MaximumIterations. MaximumIterations is set to 100 by default

Example 1: Nonlinear least-squares problem
A nonlinear least-squares problem is solved using a finite-difference Jacobian.

```csharp
using System;
using Imsl.Math;

public class NonlinLeastSquaresEx1 : NonlinLeastSquares.IFunction
{
    public void F(double[] x, double[] f)
    {
        f[0] = 10.0 * (x[1] - x[0] * x[0]);
        f[1] = 1.0 - x[0];
    }

    public static void Main(String[] args)
    {
        int m = 2;
        int n = 2;
        double[] x = new double[m];
        NonlinLeastSquares zs = new NonlinLeastSquares(m, n);
        zs.SetGuess(new double[]{ -1.2, 1.0 });
        zs.SetXscale(new double[]{1.0, 1.0});
        zs.SetFscale(new double[]{1.0, 1.0});
        x = zs.Solve(new NonlinLeastSquaresEx1());
        for (int k = 0; k < n; k++)
        {
            Console.Out.WriteLine("x[" + k + "] = " + x[k]);
        }
    }
}
```

Output
x[0] = 1
x[1] = 1
Example 2: Nonlinear least-squares problem

A nonlinear least-squares problem is solved using a user-supplied Jacobian.

```csharp
using System;
using Imsl.Math;

class NonlinLeastSquaresEx2 : NonlinLeastSquares.IJacobian
{
    public void F(double[] x, double[] f)
    {
        f[0] = 10.0 * (x[1] - x[0] * x[0]);
        f[1] = 1.0 - x[0];
    }

    public void Jacobian(double[] x, double[,] fjac)
    {
        fjac[0,0] = -20.0 * x[0];
        fjac[1,0] = 10.0;
        fjac[0,1] = -1.0;
        fjac[1,1] = 0.0;
    }

    public static void Main(string[] args)
    {
        int m = 2;
        int n = 2;
        double[] x = new double[n];
        NonlinLeastSquares zs = new NonlinLeastSquares(m, n);
        zs.SetGuess(new double[]{ -1.2, 1.0 });
        zs.SetXscale(new double[]{1.0, 1.0});
        zs.SetFscale(new double[]{1.0, 1.0});
        x = zs.Solve(new NonlinLeastSquaresEx2());
        for (int k = 0; k < n; k++)
        {
            Console.Out.WriteLine("x[" + k + "] = " + x[k]);
        }
    }
}
```

**Output**

x[0] = 1
x[1] = 1
NonlinLeastSquares.IFunction Interface

public interface Imsl.Math.NonlinLeastSquares.IFunction

Interface for the user supplied nonlinear least-squares function.

Method

F
abstract public void F(double[] x, double[] fvalue)

Description
User supplied nonlinear least-squares function.

Parameters

x – A double array containing the point at which the function is to be evaluated. The contents of this array must not be altered by this function.

fvalue – A double array which, on return, contains the function value at x.

NonlinLeastSquares.IJacobian Interface


Interface for the user supplied nonlinear least squares function and its Jacobian.

Method

Jacobian
abstract public void Jacobian(double[] x, double[,] jvalue)

Description
Jacobian of the user supplied nonlinear least squares function.

Parameters

x – A double array containing the point at which the Jacobian of the function is to be evaluated.
jvalue – A double matrix which, on return, contains the value of the Jacobian, of the function, at x.

---

**DenseLP Class**

```java
public class Imsl.Math.DenseLP
```

Solves a linear programming problem using an active set strategy.

Class DenseLP uses an active set strategy to solve linear programming problems, i.e., problems of the form

$$\min_{x \in \mathbb{R}^n} c^T x$$

subject to

$$b_l \leq Ax \leq b_u$$

$$x_l \leq x \leq x_u$$

where $c$ is the objective coefficient vector, $A$ is the coefficient matrix, and the vectors $b_l, b_u, x_l,$ and $x_u$ are the lower and upper bounds on the constraints and the variables, respectively.

If the linear constraints are infeasible an $L_1$ solution to the constraints are used as a replacement for the stated constraints. An exception is thrown but a generalized solution is computed and available using methods `GetSolution` or `GetDualSolution`. Similar comments hold for any of the three additional conditions:

1. There are multiple solutions;
2. some constraints are discarded, or
3. cycling in the algorithm is identified.

Properties

**IterationCount**

```csharp
public int IterationCount {get; }
```

**Description**

Returns the number of iterations used.

**Property Value**

An int containing the number of iterations used during the Solve() step.

**ObjectiveValue**

```csharp
public double ObjectiveValue {get; }
```

**Description**

Returns the optimal value of the objective function.

**Property Value**

A double scalar containing the optimal value of the objective function. If a solution has not been computed, `Double.NaN` is returned.

**RefinementType**

```csharp
public int RefinementType {get; set; }
```

**Description**

The type of refinement used, if any.

**Property Value**

A int specifying the type of refinement used.

**Remarks**

The possible settings are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No refinement. Always compute dual. This is the default.</td>
</tr>
<tr>
<td>1</td>
<td>Iterative refinement.</td>
</tr>
<tr>
<td>2</td>
<td>Use extended refinement. Iterate until no more progress.</td>
</tr>
</tbody>
</table>

If refinement is used, the coefficient matrices and other data are saved at the beginning of the computation. When finished this data together with the solution obtained is checked for consistency. If the discrepancy is too large, the solution process is restarted using the problem data just after processing the equalities, but with the final x values and final active set.
Constructors

**DenseLP**

public DenseLP(Imsl.Math.MPSReader mps)

**Description**
Constructor using an MPSReader object.

**Parameter**
- mps – A MPSReader specifying the Linear Programming problem.

**Exception**
- System.ArgumentException is thrown if the problem dimensions are not consistent.

**DenseLP**

public DenseLP(double[,] a, double[] b, double[] c)

**Description**
Constructor variables of type double.

**Parameters**
- a – A double matrix with coefficients of the constraints
- b – A double array containing the right-hand side of the constraints.
- c – A double array containing the coefficients of the objective function.

**Exception**
- System.ArgumentException is thrown if the dimensions of a, b.length, and c.length are not consistent.

Methods

**GetDualSolution**

public double[] GetDualSolution()

**Description**
Returns the dual solution.

**Returns**
A double array containing the dual solution of the linear programming problem.

**GetSolution**

public double[] GetSolution()
Description
Returns the solution x of the linear programming problem.

Returns
A double array containing the solution x of the linear programming problem.

SetConstraintType
public void SetConstraintType(int[] constraintType)

Description
Sets the types of general constraints in the matrix a.

Parameter
constraintType – A int array containing the types of general constraints.

Remarks
Let \( r_i = a_{i1}x_1 + \cdots + a_{in}x_n \)

<table>
<thead>
<tr>
<th>constraintType</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( r_i = b_i )</td>
</tr>
<tr>
<td>1</td>
<td>( r_i \leq b_u )</td>
</tr>
<tr>
<td>2</td>
<td>( r_i \geq b_l )</td>
</tr>
<tr>
<td>3</td>
<td>( b_l \leq r_i \leq b_u )</td>
</tr>
<tr>
<td>4</td>
<td>Ignore this constraint</td>
</tr>
</tbody>
</table>

SetLowerBound
public void SetLowerBound(double[] lowerBound)

Description
Sets the lower bound, \( x_l \) on the variables.

Parameter
lowerBound – A double array containing the lower bounds on the variables.

Remarks
If there is no lower bound on a variable, then 10e30 should be set as the lower bound. By default, lowerBound= 0.

SetUpperBound
public void SetUpperBound(double[] upperBound)

Description
Sets the upper bound, \( x_u \) on the variables.

Parameter
upperBound – A double array containing the upper bound on the variables.
Remarks
If there is no upper bound on a variable, then -10e30 should be set as the upper bound. By default there is no upper bound on a variable.

SetUpperLimit
public void SetUpperLimit(double[] upperLimit)

Description
Sets the upper limit of the constraints.

Parameter
   upperLimit – A double array containing the upper limit, \( b_u \), of the constraints that have both the lower and the upper bounds.

Solve
public void Solve()

Description
Solves the problem using an active set strategy.

Remarks
Solve must be invoked prior to any of the “get” methods.

Exceptions
   Imsl.Math.BoundsInconsistentException is thrown if the bounds are inconsistent.
   Imsl.Math.NoAcceptablePivotException is thrown if an acceptable pivot could not be found.
   Imsl.Math.ProblemUnboundedException is thrown if there is no finite solution to the problem.
   Imsl.Math.MultipleSolutionsException is thrown if the problem has multiple solutions producing essentially the same minimum.
   Imsl.Math.SomeConstraintsDiscardedException is thrown if some constraints are too linearly dependent on other active constraints.
   Imsl.Math.AllConstraintsNotSatisfiedException is thrown if some constraints are not satisfied.
   Imsl.Math.CyclingOccurringException is thrown if the algorithm appears to be cycling.

Example 1: Dense Linear Programming
The linear programming problem in the standard form

\[
\min f(x) = -x_1 - 3x_2
\]

subject to:
\[
\begin{align*}
x_1 + x_2 + x_3 &= 1.5 \\
x_1 + x_2 - x_4 &= 0.5
\end{align*}
\]
\[ x_1 + x_5 = 1.0 \]
\[ x_2 + x_6 = 1.0 \]
\[ x_i \geq 0, \text{ for } i = 1, \ldots, 6 \]

is solved.

```csharp
using System;
using Imsl.Math;

public class DenseLPEx1
{
    public static void Main(String[] args)
    {
        double[,] a = {{1.0, 1.0, 1.0, 0.0, 0.0, 0.0},
                        {1.0, 1.0, 0.0, -1.0, 0.0, 0.0},
                        {1.0, 0.0, 0.0, 0.0, 1.0, 0.0},
                        {0.0, 1.0, 0.0, 0.0, 0.0, 1.0}};
        double[] b = new double[]{1.5, 0.5, 1.0, 1.0};
        double[] c = new double[]{-1.0, -3.0, 0.0, 0.0, 0.0, 0.0};
        DenseLP zf = new DenseLP(a, b, c);
        zf.Solve();
        new PrintMatrix("Solution").Print(zf.GetSolution());
    }
}
```

**Output**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example 2: Dense Linear Programming**

The linear programming problem

\[
\min f(x) = -x_1 - 3x_2
\]

subject to:

\[ 0.5 \leq x_1 + x_2 \leq 1.5 \]
\[ 0 \leq x_1 \leq 1.0 \]
\[ 0 \leq x_2 \leq 1.0 \]

using System;
using Imsl.Math;

public class DenseLPEx2
{
    public static void Main(String[] args)
    {
        int[] constraintType = new int[] {3};
        double[] upperBound = new double[] {1.0, 1.0};
        double[,] a = {{1.0, 1.0}};
        double[] b = new double[] {0.5};
        double[] upperLimit = new double[] {1.5};
        double[] c = new double[] {-1.0, -3.0};

        DenseLP zf = new DenseLP(a, b, c);
        zf.SetUpperLimit(upperLimit);
        zf.SetConstraintType(constraintType);
        zf.SetUpperBound(upperBound);
        zf.Solve();
        new PrintMatrix("Solution").Print(zf.GetSolution());
        new PrintMatrix("Dual Solution").Print(zf.GetDualSolution());
        Console.Out.WriteLine("Optimal Value = " + zf.ObjectiveValue);
    }
}

Output
Solution
 0
 0 0.5
 1 1

Dual Solution
 0
 0 -1

Optimal Value = -3.5

Example 3: Linear Programming Maximize Example

Maximize the linear programming problem

max \( f(x) = 10x_1 + 15x_2 + 15x_3 + 13x_4 + 9x_5 \)

subject to the following set of restrictions:

100x_1 + 50x_2 + 50x_3 + 40x_4 + 120x_5 \leq 300
40x_1 + 50x_2 + 50x_3 + 15x_4 + 30x_5 \leq 40
0 \leq x_1 \leq 1.0; 0 \leq x_2 \leq 1.0; 0 \leq x_3 \leq 1.0; 0 \leq x_4 \leq 1.0; 0 \leq x_5 \leq 1.0

Since DenseLP minimizes, the sign of the objective function must be changed to compute this solution. The signs of the dual solution components and the optimal value must also be changed. Because \( x_2 \) and
\( x_3 \) are not uniquely determined within the bounds, this problem has a convex family of solutions. DenseLP issues an exception, MultipleSolutionsException. A particular solution is available and retrieved in the finally block.

```csharp
using System;
using Imsl.Math;

class DenseLPEx3
{
    public static void Main(System.String[] args)
    {
        int[] constraintType = new int[]{1, 1}; /* Ax <= b */
        double[] lowerVariableBound =
            new double[]{0.0, 0.0, 0.0, 0.0, 0.0};
        double[] upperVariableBound =
            new double[]{1.0, 1.0, 1.0, 1.0, 1.0};

        double[,] A = new double[,]{
            {100.0, 50.0, 50.0, 40.0, 120.0},
            {40.0, 50.0, 50.0, 15.0, 30.0}
        };
        /* constraint type Ax <= b */
        double[] b = new double[]{300.0, 40.0};
        double[] c = new double[]{10.0, 15.0, 15.0, 13.0, 9.0};
        /* Since DenseLP minimizes, change signs of the 
        objective coefficients. */
        double[] negC = new double[c.Length];
        for (int i = 0; i < c.Length; i++)
            negC[i] = - c[i];

        DenseLP zf = new DenseLP(A, b, negC);
        zf.SetLowerBound(lowerVariableBound);
        zf.SetConstraintType(constraintType);
        zf.SetUpperBound(upperVariableBound);

        try
        {
            zf.Solve();
        }
        catch (MultipleSolutionsException e)
        {
            /* x_2 and x_3 are not uniquely determined, expect multiple 
            * solutions. Catch the exception, but continue to print 
            * result found. */
            System.Console.Out.WriteLine(e.Message);
        }
        finally
        {
            double[] dSolution = zf.GetDualSolution();
```
/* Change the sign of the dual solution and optimal value since DenseLP minimizes. */
for (int i = 0; i < dSolution.Length; i++)
    dSolution[i] = -dSolution[i];
double optimalValue = -zf.ObjectiveValue;

new PrintMatrix("Solution").Print(zf.GetSolution());
new PrintMatrix("Dual Solution").Print(dSolution);
System.Console.Out.WriteLine(
    "Optimal Value = " + optimalValue);
}
}

Output
Multiple solutions giving essentially the same minimum exist.

Solution
 0
0 0
1 0.184987694831829
2 0.315012305168171
3 1
4 0

Dual Solution
 0
0 0
1 0.3

Optimal Value = 20.5

---

**MPSReader Class**

**public class Imsl.Math.MPSReader**

Reads a linear programming problem from an MPS file.

An MPS file defines a linear or quadratic programming problem. Linear programming problems read using this class are assumed to be of the form:

\[
\min_{x \in \mathbb{R}^n} c^T x
\]

subject to

\[
b_l \leq Ax \leq b_u
\]

\[
x_l \leq x \leq x_u
\]

where \(c\) is the objective coefficient vector, \(A\) is the coefficient matrix, and the vectors \(b_l, b_u, x_l,\) and \(x_u\) are the lower and upper bounds on the constraints and the variables, respectively.
The following table helps map this notation into use of MPSReader.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>Objective</td>
<td></td>
</tr>
<tr>
<td>$A$</td>
<td>Constraint matrix</td>
<td></td>
</tr>
<tr>
<td>$b_l$</td>
<td>Lower Range</td>
<td></td>
</tr>
<tr>
<td>$b_u$</td>
<td>Upper Range</td>
<td></td>
</tr>
<tr>
<td>$x_l$</td>
<td>Lower Bound</td>
<td></td>
</tr>
<tr>
<td>$x_u$</td>
<td>Upper Bound</td>
<td></td>
</tr>
</tbody>
</table>

If the MPS file specifies an equality constraint or bound, the corresponding lower and upper values will be exactly equal.

The problem formulation assumes that the constraints and bounds are two-sided. If a particular constraint or bound has no lower limit, then the corresponding entry in the structure is set to negative machine infinity. If the upper limit is missing, then the corresponding entry in the structure is set to positive machine infinity.

**MPS File Format**

There is some variability in the MPS format. This section describes the MPS format accepted by this reader.

An MPS file consists of a number of sections. Each section begins with a name in column 1. With the exception of the NAME section, the rest of this line is ignored. Lines with a '*' or '$' in column 1 are considered comment lines and are ignored.

The body of each section consists of lines divided into fields, as follows:

<table>
<thead>
<tr>
<th>Field Number</th>
<th>Columns</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2-3</td>
<td>Indicator</td>
</tr>
<tr>
<td>2</td>
<td>5-12</td>
<td>Name</td>
</tr>
<tr>
<td>3</td>
<td>15-22</td>
<td>Name</td>
</tr>
<tr>
<td>4</td>
<td>25-36</td>
<td>Value</td>
</tr>
<tr>
<td>5</td>
<td>40-47</td>
<td>Name</td>
</tr>
<tr>
<td>6</td>
<td>50-61</td>
<td>Value</td>
</tr>
</tbody>
</table>

The format limits MPS names to 8 characters and values to 12 characters. The names in fields 2, 3 and 5 are case sensitive. Leading and trailing blanks are ignored, but internal spaces are significant.

The sections in an MPS file are as follows:

NAME
ROWS
COLUMNS
RHS
RANGES (optional)
BOUNDS (optional)
QUADRATIC (optional)

ENDATA

Sections must occur in the above order.

MPS keywords, section names and indicator values, are case insensitive. Row, column and set names are case sensitive.

NAME Section

The NAME section contains the single line. A problem name can occur anywhere on the line after NAME and before columns 62. The problem name is truncated to 8 characters.

ROWS Section

The ROWS section defines the name and type for each row. Field 1 contains the row type and field 2 contains the row name. Row type values are not case sensitive. Row names are case sensitive. The following row types are allowed:

<table>
<thead>
<tr>
<th>Row Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Equality constraint</td>
</tr>
<tr>
<td>L</td>
<td>Less than or equal constraint</td>
</tr>
<tr>
<td>G</td>
<td>Greater than or equal constraint</td>
</tr>
<tr>
<td>N</td>
<td>Objective of a free row</td>
</tr>
</tbody>
</table>

COLUMNS Section

The COLUMNS section defines the nonzero entries in the objective and the constraint matrix. The row names here must have been defined in the ROWS section.

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Column name</td>
</tr>
<tr>
<td>3</td>
<td>Row name</td>
</tr>
<tr>
<td>4</td>
<td>Value for the entry whose row and column are given by fields 2 and 3</td>
</tr>
<tr>
<td>5</td>
<td>Row name</td>
</tr>
<tr>
<td>6</td>
<td>Value for the entry whose row and column are given by fields 2 and 5</td>
</tr>
</tbody>
</table>

Note: Fields 5 and 6 are optional.

The COLUMNS section can also contain markers. These are indicated by the name 'MARKER' (with the quotes) in field 3 and the marker type in field 4 or 5.

Marker type 'INTORG' (with the quotes) begins an integer group. The marker type 'INTEND' (with the quotes) ends this group. The variables corresponding to the columns defined within this group are required to be integer.

RHS Section

The RHS section defines the right-hand side of the constraints. An MPS file can contain more than one RHS set, distinguished by the RHS set name. The row names here must be defined in the ROWS section.
RANGES Section

The optional RANGES section defines two-sided constraints. An MPS file can contain more than one range set, distinguished by the range set name. The row names here must have been defined in the ROWS section.

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Range set name</td>
</tr>
<tr>
<td>3</td>
<td>Row name</td>
</tr>
<tr>
<td>4</td>
<td>Value for the entry whose row and column are given by fields 2 and 3</td>
</tr>
<tr>
<td>5</td>
<td>Row name</td>
</tr>
<tr>
<td>6</td>
<td>Value for the entry whose row and column are given by fields 2 and 5</td>
</tr>
</tbody>
</table>

Note: Fields 5 and 6 are optional.

Ranges change one-sided constraints, defined in the RHS section, into two-sided constraints. The two-sided constraint for row \( i \) depends on the range value, \( r_i \), defined in this section. The right-hand side value, \( b_i \), is defined in the RHS section. The two sided constraints for row \( i \) are given in the following table:

<table>
<thead>
<tr>
<th>Row Type</th>
<th>Lower Constraint</th>
<th>Upper Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>( b_i )</td>
<td>( b_i +</td>
</tr>
<tr>
<td>L</td>
<td>( b_i -</td>
<td>r_i</td>
</tr>
<tr>
<td>E</td>
<td>( b_i + \min(0, r_i) )</td>
<td>( b_i + \max(0, r_i) )</td>
</tr>
</tbody>
</table>

BOUNDS Section

The optional BOUNDS section defines bounds on the variables. By default, the bounds are \( 0 \leq x_i \leq \infty \). The bounds can also be used to indicate that a variable must be an integer.

More than one bound can be set for a single variable. For example, to set \( 2 \leq x_i \leq 6 \) use a LO bound with value 2 to set \( 2 \leq x_i \) and an UP bound with value 6 to add the condition \( x_i \leq 6 \).

An MPS file can contain more than one bounds set, distinguished by the bound set name.
### Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bounds type</td>
</tr>
<tr>
<td>2</td>
<td>Bounds set name</td>
</tr>
<tr>
<td>3</td>
<td>Column name</td>
</tr>
<tr>
<td>4</td>
<td>Value for the entry whose set and column are given by fields 2 and 3</td>
</tr>
<tr>
<td>5</td>
<td>Column name</td>
</tr>
<tr>
<td>6</td>
<td>Value for the entry whose set and column are given by fields 2 and 5</td>
</tr>
</tbody>
</table>

**Note:** Fields 5 and 6 are optional.

The bound types are as follows. Here $b_i$ are the bound values defined in this section, the $x_i$ are the variables, and $I$ is the set of integers.

<table>
<thead>
<tr>
<th>Bound Type</th>
<th>Definition</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO</td>
<td>Lower bound</td>
<td>$b_i \leq x_i$</td>
</tr>
<tr>
<td>UP</td>
<td>Upper bound</td>
<td>$x_i \leq b_i$</td>
</tr>
<tr>
<td>FX</td>
<td>Fixed Variable</td>
<td>$x_i = b_i$</td>
</tr>
<tr>
<td>FR</td>
<td>Free variable</td>
<td>$-\infty \leq x_i \leq \infty$</td>
</tr>
<tr>
<td>MI</td>
<td>Lower bound is minus infinity</td>
<td>$-\infty \leq x_i$</td>
</tr>
<tr>
<td>PL</td>
<td>Upper bound is positive infinity</td>
<td>$x_i \leq \infty$</td>
</tr>
<tr>
<td>BV</td>
<td>Binary variable (variable must be 0 or 1)</td>
<td>$x_i \in {0, 1}$</td>
</tr>
<tr>
<td>UI</td>
<td>Upper bound and integer</td>
<td>$x_i \leq b_i$ and $x_i \in I$</td>
</tr>
<tr>
<td>LI</td>
<td>Lower bound and integer</td>
<td>$b_i \leq x_i$ and $x_i \in I$</td>
</tr>
<tr>
<td>SC</td>
<td>Semicontinuous</td>
<td>0 or $b_i \leq x_i$</td>
</tr>
</tbody>
</table>

The bound type names are not case sensitive.

If the bound type is UP or UI and $b_i \leq x_i$ then the lower bound is set to $-\infty$.

**ENDATA Section**

The ENDATA section ends the MPS file.

### Fields

**BINARY_VARIABLE**

```java
public int BINARY_VARIABLE
```

**Description**

Variable must be either 0 or 1.

**CONTINUOUS_VARIABLE**

```java
public int CONTINUOUS_VARIABLE
```
**Description**

Variable is a real number.

---

**INTEGER_VARIABLE**

public int INTEGER_VARIABLE

**Description**

Variable must be an integer.

---

**Properties**

---

**Name**

virtual public string Name {get; }

**Description**

Returns the name of the MPS problem.

**Property Value**

A String containing the value of the name field.

**Remarks**

This is the value of the NAME field.

---

**NameBounds**

virtual public string NameBounds {get; set; }

**Description**

The name of the BOUNDS set.

**Property Value**

A String containing the name of the Bounds set.

**Remarks**

An MPS file can contain multiple sets of BOUNDS, but only one is retained by this reader. If not set, then the first set in the file is used.

---

**NameObjective**

virtual public string NameObjective {get; set; }

**Description**

The name of the free row containing the objective.

**Property Value**

A String containing the name of the free row containing the objective.
Remarks
An MPS file can contain free rows, but only one is retained by this reader as the objective. If not set, then the first free row in the file is used as the objective.

NameRanges
virtual public string NameRanges {get; set; }

Description
The name of the RANGES set.

Property Value
A String containing the name of the Ranges set.

Remarks
An MPS file can contain multiple sets of RANGES, but only one is retained by this reader. If not set, then the first set in the file is used.

NameRHS
virtual public string NameRHS {get; set; }

Description
The name of the RHS set used.

Property Value
A String containing the name of the RHS set used.

Remarks
An MPS file can contain multiple sets of RHS values, but only one is retained by this reader. If not set, then the first set in the file is used.

NumberOfBinaryConstraints
virtual public int NumberOfBinaryConstraints {get; }

Description
The number of binary constraints.

Property Value
An int containing the number of binary constraints.

Remarks
An binary constraint is the requirement that a variable be either 0 or 1. Binary constraints are also integer constraints.

NumberOfColumns
virtual public int NumberOfColumns {get; }

Description
The number of columns in the constraint matrix.
Property Value
An int containing the number of columns in the constraint matrix.

**NumberOfIntegerConstraints**
```csharp
virtual public int NumberOfIntegerConstraints {get; }
```

Description
The number of integer constraints.

Property Value
An int containing the number of integer constraints.

Remarks
An integer constraint is the requirement that a variable be an integer.

**NumberOfNonZeros**
```csharp
virtual public int NumberOfNonZeros {get; }
```

Description
The number of nonzeros in the constraint matrix.

Property Value
An int specifying the number of nonzeros in the constraint matrix.

**NumberOfRows**
```csharp
virtual public int NumberOfRows {get; }
```

Description
The number of rows in the constraint matrix.

Property Value
An int containing the number of rows in the constraint matrix.

**Objective**
```csharp
virtual public Imsl.Math.MPSReader.Row Objective {get; }
```

Description
The objective as a Row.

Property Value
A Row containing a representation of the objective.

**ObjectiveCoefficients**
```csharp
virtual public double[] ObjectiveCoefficients {get; }
```

Description
The coefficients of the objective row.
Property Value
A double[] containing the coefficients of the objective.

Constructor
MPSReader
public MPSReader()

Description
constructor for MPSReader

Methods
GetLowerBound
virtual public double GetLowerBound(int iVariable)

Description
Returns the lower bound for a variable.

Parameter
iVariable – An int specifying the number of the variable.

Returns
A double containing the lower bound for a variable.

GetLowerRange
virtual public double GetLowerRange(int iRow)

Description
Returns the lower range value for a constraint equation.

Parameter
iRow – An int specifying the row number of the equation.

Returns
A double containing the lower range value for a constraint equation.

GetNameColumn
virtual public string GetNameColumn(int iColumn)

Description
Returns the name of a constraint column. Constraint column names are also variable names.
Parameter
   iColumn – An int specifying the column for which a name is to be returned.

Returns
A String containing the name of a constraint column.

GetNameRow
virtual public string GetNameRow(int iRow)

Description
Returns the name of a constraint row.

Parameter
   iRow – An int specifying the row for which a name is to be returned.

Returns
A String containing the name of a constraint row.

GetRow
virtual public Imsl.Math.MPSReader.Row GetRow(int iRow)

Description
Returns a row of the constraint matrix or a free row.

Parameter
   iRow – An int specifying the number of the row that is to be returned.

Returns
A Row associated with the indicated row number, iRow.

GetRowCoefficients
virtual public double[] GetRowCoefficients(int iRow)

Description
Returns the coefficients of a row.

Parameter
   iRow – An int specifying the number of the row that is to be returned.

Returns
A double[] containing the coefficients associated with the indicated row number, iRow.

GetTypeVariable
virtual public int GetTypeVariable(int iVariable)

Description
Returns the type of a variable. The variable types are CONTINUOUS_VARIABLE, BINARY_VARIABLE or INTEGER_VARIABLE.
Parameter
   iVariable – An int specifying the number of the variable.

Returns
An int containing the variable type.

GetUpperBound
virtual public double GetUpperBound(int iVariable)

Description
Returns the upper bound for a variable.

Parameter
   iVariable – An int specifying the number of the variable.

Returns
A double containing the upper bound for a variable.

GetUpperRange
virtual public double GetUpperRange(int iRow)

Description
Returns the upper range value for a constraint equation.

Parameter
   iRow – An int specifying the row number of the equation.

Returns
A double containing the row number of the equation.

ProcessCommand
virtual protected internal string ProcessCommand(string command, string line)

Description
Process a section of the MPS file.

Parameters
   command – A String specifying the data file section to be processed.
   line – A String specifying the next line to be processed.

Returns
A String containing the next line to be processed. This line was read, but was not part of the section
being processed.

Read
virtual public void Read(System.IO.StreamReader reader)
Description
Reads and parses the MPS file.

Parameter
reader – The StreamReader that has been associated with the data file.

Example: Reading an MPS file
This example reads the data for a linear programming problem from an MPS file.

using System;
using System.IO;
using Imsl.Math;

public class MPSReaderEx1
{
    public static void Main(String[] args)
    {
        FileStream aFile = File.OpenRead("testprob.mps");
        StreamReader sr = new StreamReader(aFile);
        MPSReader mps = new MPSReader();
        mps.Read(sr);

        Console.Out.WriteLine(mps.Name);
        Console.Out.WriteLine(mps.NameRHS);
        Console.Out.WriteLine(mps.NameBounds);
        Console.Out.WriteLine(mps.NameRanges);

        int nRows = mps.NumberOfRows;
        System.Console.Out.WriteLine("NumberOfConstraints " + nRows);
        for (int i = 0; i < nRows; i++)
        {
                                         " <= row[" + i + "] = " + mps.GetNameRow(i) +
                                         " <= " + mps.GetUpperRange(i));
        }

        int nColumns = mps.NumberOfColumns;
        System.Console.Out.WriteLine("NumberOfColumns " + nColumns);
        for (int i = 0; i < nColumns; i++)
        {
                                         " <= var[" + i + "] = " + mps.GetNameColumn(i) +
                                         " <= " + mps.GetUpperBound(i));
        }

        System.Console.Out.WriteLine("NumberOfNonZeros " + mps.NumberOfNonZeros);
        for (int iRow = 0; iRow < nRows; iRow++)
        {
            System.Console.Out.WriteLine(" row " + mps.GetNameRow(iRow));
            System.Collections.IEnumerator iter = mps.GetRow(iRow).Iterator();
            while (iter.MoveNext())
            {
                MPSReader.Element elem = (MPSReader.Element) iter.Current;
            }
        }
    }
}
int iColumn = elem.Column;
System.String nameColumn = mps.GetNameColumn(iColumn);
System.Console.Out.WriteLine(" " +
nameColumn + ": " + elem.Value);
}
}

Output
TESTPROB
RHS1
BND1
NumberOfConstraints 3
-Infinity <= row[0] = LIM1 <= 5
10 <= row[1] = LIM2 <= Infinity
7 <= row[2] = MYEQN <= 7
NumberOfColumns 3
0 <= var[0] = XONE <= 4
-1 <= var[1] = YTWO <= 1
0 <= var[2] = ZTHREE <= Infinity
NumberOfNonZeros 6
row LIM1
  XONE: 1
  YTWO: 1
row LIM2
  XONE: 1
  ZTHREE: 1
row MYEQN
  YTWO: -1
  ZTHREE: 1

MPSReader.Element Class

public class Imsl.Math.MPSReader.Element
An element in the sparse contraint matrix.

Properties

Column
virtual public int Column {get; }
Description
The column index.

Property Value
An int specifying the column index.

Value
virtual public double Value {get; }

Description
The value of the element.

Property Value
A double specifying the value of the element.

MPSReader.Row Class

public class Imsl.Math.MPSReader.Row
A row either in the constraint matrix or a free row.

Properties

Coefficients
virtual public double[] Coefficients {get; }

Description
The coefficients of this row as a dense array.

Property Value
A double[] containing the coefficients of this row.

Name
virtual public string Name {get; }

Description
The name of this row.

Property Value
A String containing the name of this row.

NumberOfNonZeros
virtual public int NumberOfNonZeros {get; }

384 • MPSReader.Row  IMSL C# Numerical Library
**Description**
The number of nonzero elements in this row.

**Property Value**
An `int` containing the number of nonzero elements in this row.

**Method**

**Iterator**

```csharp
virtual public System.Collections.IEnumerator Iterator()
```

**Description**

Returns an iterator over the elements in this row.

**Returns**

An `IEnumerator` object that can be used as an iterator over the elements in this row.

**Remarks**

This is used to retrieve the coefficients in a sparse form.

See Also:  (p. 383)

---

**QuadraticProgramming Class**

```csharp
public class Imsl.Math.QuadraticProgramming
```

Solves the convex quadratic programming problem subject to equality or inequality constraints.

Class `QuadraticProgramming` is based on M.J.D. Powell’s implementation of the Goldfarb and Idnani dual quadratic programming (QP) algorithm for convex QP problems subject to general linear equality/inequality constraints (Goldfarb and Idnani 1983); i.e., problems of the form

\[
\min_{x \in \mathbb{R}^n} g^T x + \frac{1}{2} x^T H x
\]

subject to

\[
A_1 x = b_1
\]

\[
A_2 x \geq b_2
\]
given the vectors \( b_1, b_2 \), and \( g \), and the matrices \( H, A_1, \) and \( A_2 \). \( H \) is required to be positive definite. In this case, a unique \( x \) solves the problem or the constraints are inconsistent. If \( H \) is not positive definite, a positive definite perturbation of \( H \) is used in place of \( H \). For more details, see Powell (1983, 1985).

If a perturbation of \( H, H + \alpha I \), is used in the \( QP \) problem, then \( H + \alpha I \) also should be used in the definition of the Lagrange multipliers.

If the constraints are infeasible an exception is thrown. See Example 3 where the exception is caught and printed.

**Property**

**NoMoreProgress**

```csharp
public bool NoMoreProgress { get; }
```

**Description**

Contains status of `true` or `false` if computer rounding error is inhibiting improvement in the objective function.

**Property Value**

Is `true` if due to computer rounding error, a change in the variables fails to improve the objective function.

**Remarks**

Usually the solution is close to optimum.

**Constructor**

**QuadraticProgramming**

```csharp
public QuadraticProgramming(double[,] h, double[] g, double[,] aEquality, double[] bEquality, double[,] aInequality, double[] bInequality)
```

**Description**

Solve a quadratic programming problem.

**Parameters**

- \( h \) – A square array containing the Hessian. It must be positive definite.
- \( g \) – A `double` array containing the coefficients of the linear term of the objective function.
- \( aEquality \) – A rectangular matrix containing the equality constraints. It can be null if there are no equality constraints.
- \( bEquality \) – A `double` array containing the right-side of the equality constraints. It can be null if there are no equality constraints.
aInequality – A rectangular matrix containing the inequality constraints. It can be null if there are no inequality constraints.
bInequality – A double array containing the right-side of the inequality constraints. It can be null if there are no inequality constraints.

Exception
Imsl.Math.InconsistentSystemException is thrown if the problem is inconsistent.

Methods

GetDualSolution
public double[] GetDualSolution()

Description
Returns the dual (Lagrange multipliers).

Returns
A double array containing the dual.

GetSolution
public double[] GetSolution()

Description
Returns the solution.

Returns
A double array containing the unique solution.

Example 1: Solve a Quadratic Programming Problem
The quadratic programming problem is to minimize

\[ x_0^2 + x_1^2 + x_2^2 + x_3^2 + x_4^2 - 2x_1x_2 - 2x_3x_4 - 2x_0 \]

subject to

\[ x_0 + x_1 + x_2 + x_3 + x_4 = 5 \]

\[ x_2 - 2x_3 - 2x_4 = -3 \]

using System;
using Imsl.Math;
public class QuadraticProgrammingEx1
{
    public static void Main(String[] args)
    {
        double[,] h = {
            {2, 0, 0, 0, 0},
            {0, 2, -2, 0, 0},
            {0, -2, 2, 0, 0},
            {0, 0, 0, 2, -2},
            {0, 0, 0, -2, 2}
        };
        double[,] aeq = {
            {1, 1, 1, 1, 1},
            {0, 0, 1, -2, -2}
        };
        double[] beq = new double[]{5, -3};
        double[] g = new double[]{-2, 0, 0, 0, 0};
        QuadraticProgramming qp =
            new QuadraticProgramming(h, g, aeq, beq, null, null);

        // Print the solution and its dual
        new PrintMatrix("x").Print(qp.GetSolution());
        new PrintMatrix("dual").Print(qp.GetDualSolution());
    }
}

Output

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dual</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1.8329135783152E-32</td>
</tr>
</tbody>
</table>

Example 2: Solve a Quadratic Programming Problem

The quadratic programming problem is to minimize

\[ x_0^2 + x_1^2 + x_2^2 \]

subject to

\[ x_0 + 2x_1 - x_2 = 4 \]
Example 3: Solve a Quadratic Programming Problem with Inconsistent System Constraints

In the quadratic programming problem variables 2 and 6 are fixed at the value zero by the equality constraints. The inequalities propose that the sums of the variables are at least 5.1 and no more than 4.9. These last two are inconsistent conditions, causing the NoLPSolutionException to be thrown.
using Imsl.Math;

public class QuadraticProgrammingEx3
{
    public static void Main(System.String[] args)
    {
        double[,] h = {
            {2.000, 0.000, 0.000, 0.000, 0.000, 0.000},
            {0.000, 2.000, 0.000, 0.000, 0.000, 0.000},
            {0.000, 0.000, 2.000, 0.000, 0.000, 0.000},
            {0.000, 0.000, 0.000, 2.000, 0.000, 0.000},
            {0.000, 0.000, 0.000, 0.000, 2.000, 0.000},
            {0.000, 0.000, 0.000, 0.000, 0.000, 2.000}};
        double[] g = { 5.000, 5.000, 5.000, 5.000, 5.000, 5.000 };
        double[,] aEquality = {
            {0.000, 1.000, 0.000, 0.000, 0.000, 0.000},
            {0.000, 0.000, 0.000, 0.000, 0.000, 1.000}};
        double[] bEquality = { 0.000, 0.000 };
        double[,] aInequality = {
            {1.000, 1.000, 1.000, 1.000, 1.000, 1.000},
            {-1.000, -1.000, -1.000, -1.000, -1.000, -1.000}};
        double delta = 0.1; // change to 0.0 to pass
        double[] bInequality = { 5 + delta, -5 + delta };

        try
        {
            QuadraticProgramming qp = new QuadraticProgramming(h, g,
                aEquality, bEquality, aInequality, bInequality);
            double[] x = qp.GetSolution();
            new Imsl.Math.PrintMatrix("Solution").Print(x);
        }
        catch (Imsl.Math.NoLPSolutionException e)
        {
            WriteCutString(e.Message, 72);
        }
        catch (System.Exception e)
        {
            WriteCutString(e.Message, 72);
        }
    }

    public static void WriteCutString(string value, int interval)
    {
        int rem = value.Length % interval;
        int result = value.Length / interval + rem / (1 > rem ? 1 : rem);
        for (int i = 0; i < result; i++)
        {
            Console.WriteLine(value.Substring(i * interval,
                (interval < (value.Length - i * interval)) ? interval :
                    (value.Length - i * interval)));
        }
    }
}
Output

No solution for the LP problem was found. All constraints are not satisfied. L1 minimization was applied to all constraints (including bounds and simple variables) but the equalities, to approximate violated non-equalities as well as possible. If a feasible solution is possible then try using refinement.

MinConGenLin Class

public class Imsl.Math.MinConGenLin

Minimizes a general objective function subject to linear equality/inequality constraints.

The class MinConGenLin is based on M.J.D. Powell’s TOLMIN, which solves linearly constrained optimization problems, i.e., problems of the form

\[ \min f(x) \]

subject to

\[ A_1 x = b_1 \]

\[ A_2 x \leq b_2 \]

\[ x_l \leq x \leq x_u \]

given the vectors \( b_1, b_2, x_l, \) and \( x_u \) and the matrices \( A_1 \) and \( A_2 \).

The algorithm starts by checking the equality constraints for inconsistency and redundancy. If the equality constraints are consistent, the method will revise \( x^0 \), the initial guess, to satisfy

\[ A_1 x = b_1 \]
Next, $x^0$ is adjusted to satisfy the simple bounds and inequality constraints. This is done by solving a sequence of quadratic programming subproblems to minimize the sum of the constraint or bound violations.

Now, for each iteration with a feasible $x^k$, let $J_k$ be the set of indices of inequality constraints that have small residuals. Here, the simple bounds are treated as inequality constraints. Let $I_k$ be the set of indices of active constraints. The following quadratic programming problem

$$\min f(x^k) + d^T \nabla f(x^k) + \frac{1}{2} d^T B_k d$$

subject to

$$a_j d = 0, \ j \in I_k$$

$$a_j d \leq 0, \ j \in J_k$$

is solved to get $(d^k, \lambda^k)$ where $a_j$ is a row vector representing either a constraint in $A_1$ or $A_2$ or a bound constraint on $x$. In the latter case, the $a_j = e_j$ for the bound constraint $x_i \leq (x_u)$; and $a_j = -e_i$ for the constraint $-x_i \leq (x_l)$. Here, $e_i$ is a vector with 1 as the $i$-th component, and zeros elsewhere. Variables $\lambda^k$ are the Lagrange multipliers, and $B_k$ is a positive definite approximation to the second derivative $\nabla^2 f(x^k)$.

After the search direction $d^k$ is obtained, a line search is performed to locate a better point. The new point $x^{k+1} = x^k + \alpha^k d^k$ has to satisfy the conditions

$$f(x^k + \alpha^k d^k) \leq f(x^k) + 0.1 \alpha^k (d^k)^T \nabla f(x^k)$$

and

$$(d^k)^T \nabla f(x^k + \alpha^k d^k) \geq 0.7 (d^k)^T \nabla f(x^k)$$

The main idea in forming the set $J_k$ is that, if any of the equality constraints restricts the step-length $\alpha^k$, then its index is not in $J_k$. Therefore, small steps are likely to be avoided.

Finally, the second derivative approximation $B^k$, is updated by the BFGS formula, if the condition

$$(d^k)^T \nabla f(x^k + \alpha^k d^k) - \nabla f(x^k) > 0$$

holds. Let $x^k \leftarrow x^{k+1}$, and start another iteration.
The iteration repeats until the stopping criterion

\[ \| \nabla f(x^k) - A^k \lambda^k \|_2 \leq \tau \]

is satisfied. Here \( \tau \) is the supplied tolerance. For more details, see Powell (1988, 1989).

### Properties

**FinalActiveConstraintsNum**

```csharp
public int FinalActiveConstraintsNum {get; }
```

**Description**

Returns the final number of active constraints.

**Property Value**

An int scalar containing the final number of active constraints.

**NumberOfProcessors**

```csharp
public int NumberOfProcessors {get; set; }
```

**Description**

Perform the parallel calculations with the maximum possible number of processors set to

**Property Value**

An int indicating the maximum possible number of processors to use.

**Remarks**

By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**ObjectiveValue**

```csharp
public double ObjectiveValue {get; }
```

**Description**

Returns the value of the objective function.

**Property Value**

A double scalar containing the value of the objective function.

**Parallel**

```csharp
public bool Parallel {get; set; }
```
**Description**
Enable or disable performing MinConGenLin.IFunctionn.F in parallel.

**Property Value**
A bool indicating whether or not the MinConGenLin.IFunctionn.F calculations are to be performed in parallel.

**Remarks**
By default, Parallel = true. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**Tolerance**
public double Tolerance {get; set; }

**Description**
The nonnegative tolerance on the first order conditions at the calculated solution.

**Property Value**
A double scalar containing the tolerance.

**Constructor**

**MinConGenLin**

```csharp
public MinConGenLin(Imsl.Math.MinConGenLin.IFunction fcn, int nvar, int ncon, int neq, double[] a, double[] b, double[] lowerBound, double[] upperBound)
```

**Description**
Constructor for MinConGenLin.

**Parameters**
- `fcn` – The user-supplied MinConGenLin.IFunction to be minimized.
- `nvar` – An int scalar containing the number of variables.
- `ncon` – An int scalar containing the number of linear constraints (excluding simple bounds).
- `neq` – An int scalar containing the number of linear equality constraints.
- `a` – A double array containing the equality constraint gradients in the first neq rows followed by the inequality constraint gradients. `a.length = ncon * nvar`.
- `b` – A double array containing the right-hand sides of the linear constraints.
- `lowerBound` – A double array containing the lower bounds on the variables. `lowerBound.length = nvar`.
- `upperBound` – A double array containing the upper bounds on the variables. `upperBound.length = nvar`.

---

*IMSL C# Numerical Library*
Exception

System.ArgumentException is thrown if the dimensions of nvar, ncon, neq, a.length, b.length, lowerBound.length and upperBound.length are not consistent.

Methods

GetFinalActiveConstraints
public int[] GetFinalActiveConstraints()

Description
Returns the indices of the final active constraints.

Returns
An int array containing the indices of the final active constraints.

GetLagrangeMultiplierEstimate
public double[] GetLagrangeMultiplierEstimate()

Description
Returns the Lagrange multiplier estimates of the final active constraints.

Returns
A double array containing the Lagrange multiplier estimates of the final active constraints.

GetSolution
public double[] GetSolution()

Description
Returns the computed solution.

Returns
A double array containing the computed solution.

SetGuess
public void SetGuess(double[] guess)

Description
Sets an initial guess of the solution.

Parameter
   guess – A double array containing an initial guess.

Solve
public void Solve()
Description
Minimizes a general objective function subject to linear equality/inequality constraints.

Exceptions
Imsl.Math.ConstraintsInconsistentException is thrown if the constraints are inconsistent.
Imsl.Math.VarBoundsInconsistentException is thrown if the bounds on the variables are inconsistent.
Imsl.Math.ConstraintsNotSatisfiedException is thrown if a solution satisfying the constraints could not be found.
Imsl.Math.EqualityConstraintsException is thrown if the variables are determined by the constraints.

Example 1: Linear Constrained Optimization
The problem
\[
\min f(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 2x_2x_3 - 2x_4x_5 - 2x_1
\]
subject to
\[
x_1 + x_2 + x_3 + x_4 + x_5 = 5
\]
\[
x_3 - 2x_4 - 2x_5 = -3
\]
\[0 \leq x \leq 10\]
is solved.

using System;
using Imsl.Math;

public class MinConGenLinEx1 : MinConGenLin.IFunction
{
    public double F(double[] x)
    {
    }

    public static void Main(String[] args)
    {
        int neq = 2;
    }
}
Example 2: Linear Constrained Optimization

The problem

\[ \min f(x) = -x_0 x_1 x_2 \]

subject to

\[ -x_0 - 2x_1 - 2x_2 \leq 0 \]

\[ x_0 + 2x_1 + 2x_2 \leq 72 \]

\[ 0 \leq x_0 \leq 20 \]

\[ 0 \leq x_1 \leq 11 \]

\[ 0 \leq x_2 \leq 42 \]
is solved with an initial guess of $x_0 = 10$, $x_1 = 10$ and $x_2 = 10$.

using System;
using Imsl.Math;

public class MinConGenLinEx2 : MinConGenLin.IGradient
{
    public double F(double[] x)
    {
        return -x[0] * x[1] * x[2];
    }

    public void Gradient(double[] x, double[] g)
    {
        g[0] = -x[1] * x[2];
        g[1] = -x[0] * x[2];
        g[2] = -x[0] * x[1];
    }

    public static void Main(String[] args)
    {
        int neq = 0;
        int ncon = 2;
        int nvar = 3;
        double[] a = new double[]{-1.0, -2.0, -2.0, 1.0, 2.0, 2.0};
        double[] xlb = new double[]{0.0, 0.0, 0.0};
        double[] xub = new double[]{20.0, 11.0, 42.0};
        double[] b = new double[]{0.0, 72.0};

        MinConGenLin.IGradient fcn = new MinConGenLinEx2();
        MinConGenLin zf = new MinConGenLin(fcn, nvar, ncon, neq, a, b, xlb, xub);
        zf.SetGuess(new double[]{10.0, 10.0, 10.0});
        zf.Solve();
        new PrintMatrix("Solution").Print(zf.GetSolution());
        Console.Out.WriteLine("Objective value = " + zf.ObjectiveValue);
    }
}

Output

Solution

0
0 20
1 11
2 15

Objective value = -3300
**MinConGenLin.IFunction Interface**

public interface Imsl.Math.MinConGenLin.IFunction

Public interface for the user-supplied function to evaluate the function to be minimized.

**Method**

\( F \)

abstract public double F(double[] x)

**Description**

Public interface for the function to be minimized.

**Parameter**

- \( x \) – A double array, the point at which the function is evaluated. \( x.length \) equals the number of variables.

**Returns**

A double scalar, the function value at \( x \).

---

**MinConGenLin.IGradient Interface**


Public interface for the user-supplied function to compute the gradient.

**Method**

\( \text{Gradient} \)

abstract public void Gradient(double[] x, double[] g)

**Description**

Public interface for the user-supplied function to compute the gradient at point \( x \).
Parameters

- $x$ – A double array, the point at which the gradient is evaluated. $x.length$ equals the number of variables.
- $g$ – A double array which, on return, contains the values of the gradient of the objective function.

BoundedLeastSquares Class

```csharp
public class Imsl.Math.BoundedLeastSquares
{
    solves a nonlinear least-squares problem subject to bounds on the variables using a modified
    Levenberg-Marquardt algorithm.

    Class BoundedLeastSquares uses a modified Levenberg-Marquardt method and an active set strategy
to solve nonlinear least-squares problems subject to simple bounds on the variables. The problem is
stated as follows:

    \[
    \min \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2
    \]

subject to

    \[l \leq x \leq u\]

Here $m \geq n$, $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$, and $f_i(x)$ is the $i$-th component function of $F(x)$. From a given starting point,
an active set $IA$, which contains the indices of the variables at their bounds, is built. A variable is called a
"free variable" if it is not in the active set. The routine then computes the search direction for the free
variables according to the formula

\[
d = -(J^T J + \mu I)^{-1} J^T F
\]

where $\mu$ is the Levenberg-Marquardt parameter, $F = F(x)$, and $J$ is the Jacobian with respect to the free
variables. The search direction for the variables in $IA$ is set to zero. The trust region approach discussed
by Dennis and Schnabel (1983) is used to find the new point. Finally, the optimality conditions are
checked. The conditions are:

- $\|g(x_i)\| \leq \varepsilon, l_i < x_i < u_i$
- $g(x_i) < 0, x_i = u_i$
- $g(x_i) > 0, x_i = l_i$

where $\varepsilon$ is a gradient tolerance. This process is repeated until the optimality criterion is achieved.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality
condition is met for the free variables but not for all variables in $IA$, the active set. In the latter case, a
variable that violates the optimality condition will be dropped out of $IA$. For more details on the
Levenberg-Marquardt method, see Levenberg (1944) or Marquardt (1963). For more detail on the active
set strategy, see Gill and Murray (1976).
Properties

**AbsoluteTolerance**
public double AbsoluteTolerance {get; set; }

_DESCRIPTION_
The absolute function tolerance.

**Property Value**
A double scalar containing the absolute function tolerance.

**Digits**
public int Digits {get; set; }

_DESCRIPTION_
The number of good digits in the function.

**Property Value**
A int scalar containing the number of good digits.

**GradientTolerance**
public double GradientTolerance {get; set; }

_DESCRIPTION_
The scaled gradient tolerance.

**Property Value**
A double scalar containing the scaled gradient tolerance.

**MaximumFunctionEvals**
public int MaximumFunctionEvals {get; set; }

_DESCRIPTION_
The maximum number of function evaluations.

**Property Value**
A int scalar containing the maximum number of function evaluations.

**MaximumIterations**
public int MaximumIterations {get; set; }

_DESCRIPTION_
The maximum number of iterations.

**Property Value**
A int scalar containing the maximum number of iterations.

**MaximumJacobianEvals**
public int MaximumJacobianEvals {get; set; }

Optimization BoundedLeastSquares • 401
**Description**
The maximum number of Jacobian evaluations.

**Property Value**
A `int` scalar containing the maximum number of Jacobian evaluations.

---

**MaximumStepsize**
public double MaximumStepsize {get; set; }

**Description**
The maximum allowable step size.

**Property Value**
A `double` scalar containing the maximum allowable step size.

---

**NumberOfProcessors**
public int NumberOfProcessors {get; set; }

**Description**
Perform the parallel calculations with the maximum possible number of processors set to `NumberOfProcessors`.

**Property Value**
An `int` indicating the maximum possible number of processors to use.

**Remarks**
By default, `NumberOfProcessors` = `Environment.ProcessorCount`. If `NumberOfProcessors` is set to a number less than 1 or greater than `Environment.ProcessorCount`, `Environment.ProcessorCount` will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

---

**Parallel**
public bool Parallel {get; set; }

**Description**
Enable or disable performing MinUnconMultiVar.IFunction.F in parallel.

**Property Value**
A `bool` indicating whether or not the MinUnconMultiVar.IFunction.F calculations are to be performed in parallel.

**Remarks**
By default, `Parallel` = `true`. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

---

**RelativeTolerance**
public double RelativeTolerance {get; set; }
Description
The relative function tolerance.

Property Value
A double scalar containing the relative function tolerance.

ScaledStepTolerance
public double ScaledStepTolerance {get; set; }

Description
The scaled step tolerance.

Property Value
A double scalar containing the scaled step tolerance.

TrustRegion
public double TrustRegion {get; set; }

Description
The size of initial trust region radius.

Property Value
A double scalar containing the initial trust region radius.

Constructor

BoundedLeastSquares
public BoundedLeastSquares(Imsl.Math.BoundedLeastSquares.IFunction f, int mFunctions, int nVariables, int boundType, double[] lowerBound, double[] upperBound)

Description
Constructor for BoundedLeastSquares.

Parameters
f – The user-supplied BoundedLeastSquares.IFunction to be minimized.
mFunctions – A int scalar containing the number of functions.
nVariables – A int scalar containing the number of variables.
boundType – A int scalar containing the types of bounds on the variable.

<table>
<thead>
<tr>
<th>boundType</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>User will supply all the bounds.</td>
</tr>
<tr>
<td>1</td>
<td>All variables are nonnegative.</td>
</tr>
<tr>
<td>2</td>
<td>All variables are nonpositive.</td>
</tr>
<tr>
<td>3</td>
<td>User supplies only the bounds on first variable, all other variables will have the same bounds.</td>
</tr>
</tbody>
</table>
lowerBound – A double array containing the lower bounds on the variables.
upperBound – A double array containing the upper bounds on the variables.

Exception
System.ArgumentException is thrown if the dimensions of mFunctions, nVariables,
boundType, lowerBound.length and upperBound.length are not consistent

Methods

GetJacobianSolution
public double[,] GetJacobianSolution()

Description
Returns the Jacobian at the approximate solution.

Returns
A mFunctions x nVariables double matrix containing the Jacobian at the approximate solution.

GetResiduals
public double[] GetResiduals()

Description
Returns the residuals at the approximate solution.

Returns
A double array containing the residuals at the approximate solution.

GetSolution
public double[] GetSolution()

Description
Returns the solution.

Returns
A double array containing the computed solution.

SetFscale
public void SetFscale(double[] fscale)

Description
Sets the diagonal scaling matrix for the functions.

Parameter
fscale – A double array containing the diagonal scaling for the functions.
Remarks
The i-th component of fscale is a positive scalar specifying the reciprocal magnitude of the i-th component function of the problem. By default, fscale[] = {1}.

SetGuess
public void SetGuess(double[] guess)

Description
Sets the initial guess of the solution.

Parameter
    guess – A double array containing an initial guess.

SetInternalScale
public void SetInternalScale()

Description
The internal variable scaling option.

Remarks
With this option, the values for xscale are set internally.

SetXscale
public void SetXscale(double[] xscale)

Description
The scaling vector for the variables.

Parameter
    xscale – A double array containing the scaling vector for the variables.

Remarks
Argument xscale is used mainly in scaling the gradient and the distance between two points. See GradientTolerance and ScaledStepTolerance for more details. By default, xscale[] = {1}.

Solve
public void Solve()

Description
Solves a nonlinear least-squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm.

Exception
    Imsl.Math.FalseConvergenceException is thrown if there is a problem with convergence.
Example 1: Bounded Least Squares

The nonlinear least-squares problem

\[
\begin{align*}
\min & \quad \frac{1}{2} \sum_{i=0}^{1} f_i(x)^2 \\
\text{subject to} & \quad -2 \leq x_0 \leq 0.5 \\
& \quad -1 \leq x_1 \leq 2
\end{align*}
\]

where

\[
\begin{align*}
f_0(x) &= 10(x_1 - x_0^2) \quad \text{and} \quad f_1(x) = (1 - x_0)
\end{align*}
\]

is solved.

```csharp
using System;
using Imsl.Math;

public class BoundedLeastSquaresEx1 : BoundedLeastSquares.IFunction
{
    public void F(double[] x, double[] f)
    {
        f[0] = 10.0 * (x[1] - x[0] * x[0]);
        f[1] = 1.0 - x[0];
    }

    public static void Main(String[] args)
    {
        int m = 2;
        int n = 2;
        int ibtype = 0;
        double[] xlb = new double[] { -2.0, -1.0 };
        double[] xub = new double[] { 0.5, 2.0 };

        BoundedLeastSquares.IFunction rosbck =
            new BoundedLeastSquaresEx1();
        BoundedLeastSquares zf =
            new BoundedLeastSquares(rosbck, m, n, ibtype, xlb, xub);
        zf.Solve();
        new PrintMatrix("Solution").Print(zf.GetSolution());
    }
}
```

Output

```
Solution
0
```
Example 2: Bounded Least Squares

The nonlinear least-squares problem

\[
\min \frac{1}{2} \sum_{i=0}^{1} f_i(x)^2
\]

\[-2 \leq x_0 \leq 0.5\]

\[-1 \leq x_1 \leq 2\]

where

\[f_0(x) = 10(x_1 - x_0^2)\] and \[f_1(x) = (1 - x_0)\]

is solved. An initial guess (-1.2, 1.0) is supplied, as well as the analytic Jacobian. The residual at the approximate solution is returned.

```csharp
using System;
using Imsl.Math;

public class BoundedLeastSquaresEx2 : BoundedLeastSquares.IJacobian
{
    public void F(double[] x, double[] f)
    {
        f[0] = 10.0 * (x[1] - x[0] * x[0]);
        f[1] = 1.0 - x[0];
    }

    public void Jacobian(double[] x, double[] fjac)
    {
        fjac[0] = -20.0 * x[0];
        fjac[1] = 10.0;
        fjac[2] = -1.0;
        fjac[3] = 0.0;
    }
}

double[] xlb = { -2.0, -1.0 };
double[] xub = { -2.0, -1.0 };
```
BoundedLeastSquares.IJacobian rosbck =
    new BoundedLeastSquaresEx2();
BoundedLeastSquares zf =
    new BoundedLeastSquares(rosbck, m, n, ibtype, xlb, xub);
zf.SetGuess(new double[]{ -1.2, 1.0 });
zf.Solve();
new PrintMatrix("Solution").Print(zf.GetSolution());
new PrintMatrix("Residuals").Print(zf.GetResiduals());

Output
Solution
  0
  0  0.5
  1  0.25
Residuals
  0
  0  0
  1  0.5

BoundedLeastSquares.IFunction Interface

public interface Imsl.Math.BoundedLeastSquares.IFunction
Public interface for the user-supplied function to evaluate the function that defines the least-squares problem.

Method

F
abstract public void F(double[] x, double[] fvalue)

Description
Public interface for the user-supplied function to evaluate the function that defines the least-squares problem.

Parameters
    x – A double array, the point at which the function is to evaluated. x.length = nVariables.
    fvalue – A double array, the function values at point x. f.Length = mFunctions.
BoundedLeastSquares.IJacobian Interface


Public interface for the user-supplied function to compute the Jacobian.

Method

Jacobian

abstract public void Jacobian(double[] x, double[] fjac)

Description

Public interface for the user-supplied function to compute the Jacobian.

Parameters

  x – A double array, the point at which the Jacobian is to evaluated. x.length = nVariables.

  fjac – A double array which, on return, contains the computed Jacobian at the point x.

  fjac.length = mFunctions x nVariables.

BoundedVariableLeastSquares Class

public class Imsl.Math.BoundedVariableLeastSquares

Solve a linear least-squares problem with bounds on the variables.

BoundedVariableLeastSquares solves the least-squares problem

\[ \min_x \|Ax - b\|^2 \]

subject to the conditions

\[ \alpha_k \leq x_k \leq \beta_k \]

for all k.

This algorithm is a generalization of Imsl.Math.NonNegativeLeastSquares (p. 413), that solves the least-squares problem, \( Ax = b \), subject to all \( x_j \geq 0 \). NonNegativeLeastSquares is based on the subroutine NNLS which appeared in Lawson and Hanson (1974). The additional work on bounded variable least squares was published in a later reprint (Lawson and Hanson, 1995).
Properties

**Iterations**
public int Iterations {get; }

**Description**
The number of iterations used to find the solution.

**Property Value**
An int containing the number of iterations.

**MaxIterations**
public int MaxIterations {get; set; }

**Description**
The maximum number of iterations.

**Property Value**
An int containing the maximum number of iterations.
Default: MaxIterations = 3*a.GetLength(1).

**ResidualNorm**
virtual public double ResidualNorm {get; }

**Description**
The euclidean norm of the residual vector, $\|Ax - b\|^2$.

**Property Value**
A double containing the euclidean norm of the residual vector.

**Tolerance**
public double Tolerance {get; set; }

**Description**
The internal tolerance used to determine the relative linear dependence of a column vector for a variable moved from its initial value.

**Property Value**
A double value specifying the tolerance.
Default: Tolerance = 1.0e-7.

Constructor

**BoundedVariableLeastSquares**
public BoundedVariableLeastSquares(double[,] a, double[] b, double[] lowerBound, double[] upperBound)
Description
Construct a new BoundedVariableLeastSquares instance to solve $Ax-b$ subject to bounds on the variables. Each upper bound must be greater than or equal to the corresponding lower bound.

Parameters
- $a$ – The double input matrix.
- $b$ – A double array of length $a.GetLength(0)$.
- $\text{lowerBound}$ – A double array of length $a.GetLength(0)$ containing lower bounds. Use $\text{Double.NEGATIVE_INFINITY}$ for variables which are not bounded below.
- $\text{upperBound}$ – A double array of length $a.GetLength(0)$ containing upper bounds. Use $\text{Double.POSITIVE_INFINITY}$ for variables which are not bounded above.

Methods

GetDualSolution

```java
public double[] GetDualSolution()
```

Description
Returns the dual solution vector, $w$.

Returns
A double array containing the dual solution vector, $w$.

Remarks
If $x_j$ is at neither its upper nor lower bound then $w_j = 0$. If $x_j$ is at its lower bound then $w_j \leq 0$. If $x_j$ is at its upper bound then $w_j \geq 0$. If the upper and lower bound for the $j$-th variable are equal, fixing the value of $x_j$, then the value of $w_j$ is arbitrary.

GetSolution

```java
public double[] GetSolution()
```

Description
Returns the solution to the problem.

Returns
A double array containing the solution.

Solve

```java
virtual public void Solve()
```

Description
Find the solution $x$ to the problem for the current constraints.
Example 1: Bounded Variable Least Squares

The following example solves a linear least squares problem with bounds on the variables and compares the result to its unbounded solution. The normal of the residuals is 0.0 for the exact solution.

```csharp
using System;
using Imsl.Math;

public class BoundedVariableLeastSquaresEx1
{
    public static void Main(String[] args)
    {
        double[,] a = {{1, - 3, 2}, {- 3, 10, - 5}, {2, - 5, 6}};
        double[] b = {27, - 78, 64};
        double[] xlb = {- 1, - 1, - 1};
        double[] xub = {5, 5, 5};
        double[] xl_ub = {
            Double.NegativeInfinity,
            Double.NegativeInfinity,
            Double.NegativeInfinity
        };
        double[] xu_ub = {
            Double.PositiveInfinity,
            Double.PositiveInfinity,
            Double.PositiveInfinity
        };

        // compute the bounded solution
        BoundedVariableLeastSquares bvls = 
            new BoundedVariableLeastSquares(a, b, xlb, xub);
        bvls.Solve();
        double[] x_bounded = bvls.GetSolution();
        new PrintMatrix("Bounded Solution").Print(x_bounded);
        Console.WriteLine("Norm of the Residuals = "+
            bvls.ResidualNorm);

        // compute the unbounded solution
        bvls = new BoundedVariableLeastSquares(a, b, xl_ub, xu_ub);
        bvls.Solve();
        double[] x_unbounded = bvls.GetSolution();
        new PrintMatrix("Unbounded Solution").Print(x_unbounded);
        Console.WriteLine("Norm of the Residuals = "+
            bvls.ResidualNorm);
    }
}
```

Output

Bounded Solution

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Norm of the Residuals = 35.0142828000232
NonNegativeLeastSquares Class

public class Imsl.Math.NonNegativeLeastSquares
Solves a linear least squares problem with nonnegativity constraints.
NonNegativeLeastSquares solves the problem
\[ \min_x \|Ax - b\|^2 \]
subject to the condition \( x \geq 0 \).
If a starting point \( x_0 \) is provided, those entries of \( x_0 \) that are \( > 0 \) are first combined with a descent gradient component. The start point is the origin. When \( x_0 \) is not provided the algorithm uses only the gradient to verify that an optimum has been found. The algorithm completes using only the gradient components to reach an optimum. For more information, see Lawson and Hanson (1974).

See Also
Imsl.Math.BoundedVariableLeastSquares (p. 409)

Properties

DualTolerance
virtual public double DualTolerance {get; set; }
Description
The dual tolerance controlling when the computation stops.

Property Value
A double containing the dual tolerance. The computation stops if the largest gradient is smaller than this.
Default: DualTolerance = 0.

Optimization NonNegativeLeastSquares • 413
Iterations
virtual public int Iterations {get; }

Description
The number of iterations used to find the solution.

Property Value
An int containing the number of iterations.

MaximumTime
virtual public long MaximumTime {get; set; }

Description
The maximum time allowed for the solve step.

Property Value
A long value specifying the maximum time, in milliseconds, to be allowed for the solve step.
Default: There is no time limit.

Remarks
If MaximumTime is less than or equal to zero, then no time limit is imposed.

MaxIterations
virtual public int MaxIterations {get; set; }

Description
The maximum number of iterations.

Property Value
An int specifying the maximum number of iterations.
Default: MaxIterations = 3*a.GetLength(1).

NormTolerance
virtual public double NormTolerance {get; set; }

Description
The residual norm tolerance.

Property Value
A double containing the residual norm tolerance. The computation stops if $||rnorm||^2 \leq \text{NormTolerance} \times ||b||$, where $rnorm$ is the residual norm.
Default: NormTolerance = 0.

RankTolerance
virtual public double RankTolerance {get; set; }

Description
The tolerance used for the incoming column rank deficient check.
**Property Value**
A double value used to check for rank deficiency.
Default: RankTolerance = 2.220e-016.

**ResidualNorm**
virtual public double ResidualNorm {get; }

**Description**
The euclidean norm of the residual vector, \( \|Ax - b\|^2 \).

**Property Value**
A double containing the euclidean norm of the residual vector.

**Constructor**

**NonNegativeLeastSquares**
public NonNegativeLeastSquares(double[,] a, double[] b)

**Description**
Construct a new NonNegativeLeastSquares instance to solve \( Ax - b \) where \( x \) is a vector of \( n \) unknowns.

**Parameters**
- \( a \) – The double input matrix.
- \( b \) – A double array of length \( a.GetLength(0) \).

**Methods**

**GetDualSolution**
virtual public double[] GetDualSolution()

**Description**
Returns the dual solution vector, \( w \). If \( x_j = 0 \) then \( w_j \leq 0 \), otherwise \( w_j = 0 \).

**Returns**
A double array containing the dual solution vector, \( w \).

**GetSolution**
virtual public double[] GetSolution()

**Description**
Returns the solution to the problem, \( x \).
Returns
A double array containing the solution.

SetGuess
virtual public void SetGuess(double[] guess)

Description
Sets the initial guess.

Parameter

guess – A double array containing the initial guess.

Remarks
If set, the guess is used in a two-phase algorithm where the positive components are matched with positive gradients to choose an incoming column. If not set, the algorithm uses only the gradient to verify that an optimum has been found.

Solve
virtual public void Solve()

Description
Finds the solution to the problem for the current constraints.

Exceptions

Imsl.Math.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.

Imsl.Math.TooMuchTimeException is thrown if the maximum time allowed for the Solve method is exceeded.

Example 1: Non-negative Least Squares

Consider the following problem:

\[
\begin{bmatrix}
1 & -3 & 2 \\
-3 & 10 & -5 \\
2 & -5 & 6
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
= 
\begin{bmatrix}
27 \\
-78 \\
64
\end{bmatrix}
\]

Subject to the constraint \(x \geq 0\). The NonNegativeLeastSquares class is used to compute a solution, which is compared to the exact solution of \(\{1, -4, 7\}\).

```csharp
using System;
using Imsl.Math;
public class NonNegativeLeastSquaresEx1
{
    public static void Main(String[] args)
    {
        double[,] a = {{1, - 3, 2}, {- 3, 10, - 5}, {2, - 5, 6}};
        double[] b = {27, - 78, 64};
        NonNegativeLeastSquares nnls =
```

416 • NonNegativeLeastSquares

IMSL C# Numerical Library
```java
new NonNegativeLeastSquares(a, b);
nlsl.Solve();
double[] x = nlsl.GetSolution();
new PrintMatrix("Solution").Print(x);

// compare solution with exact answer
double[,] compare = new double[2, 3];
double[] tmp1 = Matrix.Multiply(a, x);
double[] tmp2 = Matrix.Multiply(a, new double[]{1, -4, 7});
for (int i = 0; i < compare.GetLength(1); i++)
{
    compare[0, i] = tmp1[i];
    compare[1, i] = tmp2[i];
}

PrintMatrixFormat pmf = new PrintMatrixFormat();
pmf.SetColumnLabels(new String[] {"x >= 0", "exact");
PrintMatrix pm = new PrintMatrix("Comparison of 'b'");
pm.Print(pmf, Matrix.Transpose(compare));
}
```

Output

Solution

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>18.4492753623188</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4.5072463768116</td>
</tr>
</tbody>
</table>

Comparison of 'b'

<table>
<thead>
<tr>
<th></th>
<th>x &gt;= 0</th>
<th>exact</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>27.463768115942</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-77.8840579710145</td>
<td>-78</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>63.9420289855073</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

**MinConNLP Class**

**public class Imsl.Math.MinConNLP**

General nonlinear programming solver.

MinConNLP is based on the FORTRAN subroutine, DONLP2, by Peter Spellucci and licensed from TU Darmstadt. MinConNLP uses a sequential equality constrained quadratic programming method with an active set technique, and an alternative usage of a fully regularized mixed constrained subproblem in case of nonregular constraints (i.e. linear dependent gradients in the “working sets”). It uses a slightly modified version of the Pantoja-Mayne update for the Hessian of the Lagrangian, variable dual scaling.
and an improved Armijo-type stepsize algorithm. Bounds on the variables are treated in a
gradient-projection like fashion. Details may be found in the following two papers:

P. Spellucci: *An SQP method for general nonlinear programs using only equality constrained


The problem is stated as follows:

\[
\min_{x \in \mathbb{R}^n} f(x)
\]

subject to

\[
\begin{align*}
g_j(x) &= 0, \text{for } j = 1, \ldots, m_e \\
g_j(x) &\geq 0, \text{for } j = m_e + 1, \ldots, m \\
x_l \leq x \leq x_u
\end{align*}
\]

where all problem functions are assumed to be continuously differentiable. Although default values are
provided for optional input arguments, it may be necessary to adjust these values for some problems.

Through the use of member functions, MinConNLP allows for several parameters of the algorithm to be
adjusted to account for specific characteristics of problems. The DONLP2 Users Guide provides detailed
descriptions of these parameters as well as strategies for maximizing the performance of the algorithm.

In addition, the following are a number of guidelines to consider when using MinConNLP:

- A good initial starting point is very problem specific and should be provided by the calling
  program whenever possible. See method SetGuess.

- Gradient approximation methods can have an effect on the success of MinConNLP. Selecting a
  higher order approximation method may be necessary for some problems. See property
  DifferentiationType.

- If a two sided constraint \( l_i \leq g_i(x) \leq u_i \) is transformed into two constraints, \( g_{2i}(x) \geq 0 \)
  and \( g_{2i+1}(x) \geq 0 \), then choose BindingThreshold \(< 1/2(u_i - l_i)/max\{1, \|\nabla g_i(x)\|\}\), or at least try to
  provide an estimate for that value. This will increase the efficiency of the algorithm. See property
  BindingThreshold.

- The parameter ierr provided in the interface to the user supplied function F can be very useful in
  cases when evaluation is requested at a point that is not possible or reasonable. For example, if
  evaluation at the requested point would result in a floating point exception, then setting ierr to
  true and returning without performing the evaluation will avoid the exception. MinConNLP will
  then reduce the stepsize and try the step again. Note, if ierr is set to true for the initial guess,
  then an error is issued.

**Properties**

**BindingThreshold**

public double BindingThreshold {get; set; }
Description
The binding threshold for constraints.

Property Value
A double scalar value specifying the binding threshold for constraints.

Remarks
In the initial phase of minimization a constraint is considered binding if
\[
g_i(x) \leq \frac{g_i(x)}{\max(1, \|\nabla g_i(x)\|)} \cdot \text{BindingThreshold} \quad i = M_e + 1, \ldots, M
\]
Good values are between .01 and 1.0. If BindingThreshold is chosen too small then identification of the correct set of binding constraints may be delayed. Contrary, if BindingThreshold is too large, then the method will often escape to the full regularized SQP method, using individual slack variables for any active constraint, which is quite costly. For well scaled problems BindingThreshold = 1.0 is reasonable. By default, BindingThreshold is set to .5 * PenaltyBound.

Exception
System.ArgumentException is thrown if BindingThreshold is less than or equal to 0.0

BoundViolationBound
public double BoundViolationBound {get; set; }

Description
The amount by which bounds may be violated during numerical differentiation.

Property Value
A double scalar value specifying the amount by which bounds may be violated during numerical differentiation.

Remarks
By default, BoundViolationBound is set to 1.0.

Exception
System.ArgumentException is thrown if BoundViolationBound is set to a value less than or equal to 0.0

DifferentiationType
public int DifferentiationType {get; set; }

Description
The type of numerical differentiation to be used.

Property Value
An int scalar value specifying the type of numerical differentiation to be used. By default, DifferentiationType is set to 1.
<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Use a forward difference quotient with discretization stepsize $0.1 \left( \frac{1}{2} \right)^{1/2}$ componentwise relative. This is the default value used.</td>
</tr>
<tr>
<td>2</td>
<td>Use the symmetric difference quotient with discretization stepsize $0.1 \left( \frac{1}{3} \right)^{1/3}$ componentwise relative.</td>
</tr>
<tr>
<td>3</td>
<td>Use the sixth order approximation computing a Richardson extrapolation of three symmetric difference quotient values. This uses a discretization stepsize $0.01 \left( \frac{1}{7} \right)^{1/7}$.</td>
</tr>
</tbody>
</table>

**Exception**

System.ArgumentException is thrown if DifferentiationType is set to a value less than or equal to 0, or greater than or equal to 4.

### FunctionPrecision

```csharp
public double FunctionPrecision { get; set; }
```

**Description**

The relative precision of the function evaluation routine.

**Property Value**

A double scalar value specifying the relative precision of the function evaluation routine.

**Remarks**

By default, FunctionPrecision is set to 2.2e-16.

**Exception**

System.ArgumentException is thrown if FunctionPrecision is set to a value less than or equal to 0.0.

### GradientPrecision

```csharp
public double GradientPrecision { get; set; }
```

**Description**

The relative precision in gradients.

**Property Value**

A double scalar value specifying the relative precision in gradients.

**Remarks**

By default, GradientPrecision is set to 2.2e-16.

**Exception**

System.ArgumentException is thrown if less than or equal to 0.0.

### MaximumIterations

```csharp
public int MaximumIterations { get; set; }
```
Description
The maximum number of iterations allowed.

Property Value
An int specifying the maximum number of iterations allowed.

Remarks
By default, MaximumIterations is set to 200.

Exception
System.ArgumentException is thrown if MaximumIterations is less than or equal to 0

MultiplierError
public double MultiplierError {get; set; }

Description
The error allowed in the multipliers.

Property Value
A double scalar value specifying the error allowed in the multipliers.

Remarks
A negative multiplier of an inequality constraint is accepted (as zero) if its absolute value is less than MultiplierError. By default, MultiplierError is set to $e^{2\log\varepsilon}/3$.

Exception
System.ArgumentException is thrown if MultiplierError is less than or equal to 0.0

NumberOfProcessors
public int NumberOfProcessors {get; set; }

Description
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

Property Value
An int indicating the maximum possible number of processors to use.

Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

Parallel
public bool Parallel {get; set; }

Description
Enable or disable performing MinConNLP.IFunction.F in parallel.
**Property Value**
A bool indicating whether or not the MinConNLP.IFunction.F calculations are to be performed in parallel.

**Remarks**
By default, Parallel = true. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

---

**PenaltyBound**
public double PenaltyBound {get; set; }

**Description**
The universal bound for describing how much the unscaled penalty-term may deviate from zero.

**Property Value**
A double scalar value specifying the universal bound for describing how much the unscaled penalty-term may deviate from zero.

**Remarks**
A small PenaltyBound diminishes the efficiency of the solver because the iterates then will follow the boundary of the feasible set closely. Conversely, a large PenaltyBound may degrade the reliability of the code. By default, PenaltyBound is set to 1.0.

**Exception**
System.ArgumentException is thrown if PenaltyBound is less than or equal to 0.0

---

**ScalingBound**
public double ScalingBound {get; set; }

**Description**
The scaling bound for the internal automatic scaling of the objective function.

**Property Value**
A double scalar value specifying the scaling variable for the problem function.

**Remarks**
By default, ScalingBound is set to 1.0e4.

**Exception**
System.ArgumentException is thrown if ScalingBound is less than or equal to 0.0

---

**ViolationBound**
public double ViolationBound {get; set; }

**Description**
Defines allowable constraint violations of the final accepted result.

**Property Value**
A double scalar value specifying the allowable constraint violations of the final accepted result.
Remarks
Constraints are satisfied if \( |g_i(x)| \leq \text{ViolationBound} \), and \( g_i(x) \geq -\text{ViolationBound} \) respectively. By default, \( \text{ViolationBound} \) is set to \( \min(\text{BindingThreshold}/10, \max(\epsilon_{\text{diff}}, \min(\text{BindingThreshold}/10, \max((1.0 - 6)\text{BindingThreshold}, \text{small}_w)))) \).

Exception
System.ArgumentException is thrown if \( \text{ViolationBound} \) is less than or equal to 0.0

Constructor

**MinConNLP**

```csharp
public MinConNLP(int mTotalConstraints, int mEqualityConstraints, int nVariables)
```

Description
Nonlinear programming solver constructor.

Parameters
- \( mTotalConstraints \) – An int scalar value which defines the total number of constraints.
- \( mEqualityConstraints \) – An int scalar value which defines the number of equality constraints.
- \( nVariables \) – An int scalar value which defines the number of variables.

Methods

GetConstraintResiduals
```csharp
public double[] GetConstraintResiduals()
```

Description
Returns the constraint residuals.

Returns
A double array containing the constraint residuals.

GetLagrangeMultiplierEst
```csharp
public double[] GetLagrangeMultiplierEst()
```

Description
Returns the Lagrange multiplier estimates of the constraints.
Returns
A double array containing the Lagrange multiplier estimates of the constraints.

GetSolution
public double[] GetSolution()

Description
Returns the last computed solution.

Returns
A double array containing the solution.

Remarks
This is the same solution as returned by the solve method.

SetGuess
public void SetGuess(double[] guess)

Description
Sets the initial guess of the minimum point of the input function.

Parameter
    guess – A double array specifying the initial guess of the minimum point of the input function.

Remarks
By default, the elements of this array are set to x, (with the smallest value of ||x||₂) that satisfies the bounds.

SetXlowerBound
public void SetXlowerBound(double[] lower)

Description
Sets the lower bounds on the variables.

Parameter
    lower – A double array specifying the lower bounds on the variables.

Remarks
By default, the elements of this array are set to -1.79e308.

SetXscale
public void SetXscale(double[] scale)

Description
The internal scaling of the variables.

Parameter
    scale – A double array specifying the internal scaling of the variables.
Remarks
The initial value given and the objective function and gradient evaluations, however, are always given in
the original unscaled variables. The first internal variable is obtained by dividing the values \( x[i] \) by
\( Xscale[i] \). By default, \( Xscale[i] \) is set to 1.0.

Exception
System.ArgumentException is thrown if \( Xscale[i] \) is less than or equal to 0.0

SetXupperBound
public void SetXupperBound(double[] upper)

Description
Sets the upper bounds on the variables.

Parameter
upper – A double array specifying the upper bounds on the variables.

Remarks
By default, the elements of this array are set to 1.79e308.

Solve
public double[] Solve(Imsl.Math.MinConNLP.IFunction f)

Description
Solve a general nonlinear programming problem using the successive quadratic programming algorithm
with a finite-difference gradient or with a user-supplied gradient.

Parameter
f – Defines the user-supplied MinConNLP.IFunction to be evaluated at a given point. f can be
used to supply a MinConNLP.IGradient of the function. If f implements IGradient the
user-supplied gradient is used. Otherwise, an attempt to solve the problem is made using a
finite-difference gradient.

Returns
A double array containing the last computed solution of the nonlinear programming problem.

Exceptions
Imsl.Math.ConstraintEvaluationException is thrown if a constraint evaluation returns an
error.
Imsl.Math.ObjectiveEvaluationException is thrown if objective evaluation returns an error.
Imsl.Math.WorkingSetSingularException is thrown if
Imsl.Math.QPInfeasibleException is thrown if the working set is singular in dual extended
QP.
Imsl.Math.PenaltyFunctionPointInfeasibleException is thrown if the penalty function
point infeasible.
Imsl.Math.LimitingAccuracyException is thrown if limiting accuracy reached for a singular problem.
Imsl.Math.TooManyIterationsException is thrown if maximum number of iterations exceeded.
Imsl.Math.NoAcceptableStepsizeException is thrown if there is no acceptable stepsize.
Imsl.Math.BadInitialGuessException is thrown if the penalty function point infeasible for original problem.
Imsl.Math.IllConditionedException is thrown if the problem is singular or ill-conditioned.
Imsl.Math.SingularException is thrown if the problem is singular.
Imsl.Math.LinearlyDependentGradientsException is thrown if the working set gradients are linearly dependent.
Imsl.Math.TerminationCriteriaNotSatisfiedException is thrown if termination criteria are not satisfied.

Example 1: Solving a general nonlinear programming problem

A general nonlinear programming problem is solved using a finite difference gradient. The problem

$$\min F(x) = (x_1 - 2)^2 + (x_2 - 1)^2$$

subject to

$$g_1(x) = x_1 - 2x_2 + 1 = 0$$

$$g_2(x) = -x_1^2/4 - x_2^2 + 1 \geq 0$$

is solved.

using System;
using Imsl.Math;

public class MinConNLPEx1 : MinConNLP.IFunction
{
    public double F(double[] x, int iact, bool[] ierr)
    {
        double result;
        ierr[0] = false;
        if (iact == 0)
        {
            result = (x[0] - 2.0) * (x[0] - 2.0) +
                     (x[1] - 1.0) * (x[1] - 1.0);
            return result;
        }
    }
}
else
{
    switch (iact)
    {
        case 1:
            result = (x[0] - 2.0 * x[1] + 1.0);
            return result;

        case 2:
            result = (-(x[0] * x[0]) / 4.0 - (x[1] * x[1])
                                   + 1.0);
            return result;

        default:
            ierr[0] = true;
            return 0.0;
    }
}

public static void Main(String[] args)
{
    int m = 2;
    int me = 1;
    int n = 2;
    MinConNLP minconnon = new MinConNLP(m, me, n);
    minconnon.SetGuess(new double[]{2.0, 2.0});
    double[] x = minconnon.Solve(new MinConNLPEx1());
    new PrintMatrix("x").Print(x);
}

Output

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.822875655532512</td>
</tr>
<tr>
<td>1</td>
<td>0.911437827766256</td>
</tr>
</tbody>
</table>

Example 2: Solving a general nonlinear programming problem

A general nonlinear programming problem is solved using a user-supplied gradient. The problem

\[
\min F(x) = (x_1 - 2)^2 + (x_2 - 1)^2
\]

subject to
\[ g_1(x) = x_1 - 2x_2 + 1 = 0 \]

\[ g_2(x) = -\frac{x_1^2}{4} - x_2^2 + 1 \geq 0 \]

is solved.

```csharp
using System;
using Imsl.Math;

public class MinConNLPEx2 : MinConNLP.IGradient
{
    public double F(double[] x, int iact, bool[] ierr)
    {
        double result;
        ierr[0] = false;
        if (iact == 0)
        {
            result = (x[0] - 2.0) * (x[0] - 2.0) +
                      (x[1] - 1.0) * (x[1] - 1.0);
            return result;
        }
        else
        {
            switch (iact)
            {
                case 1:
                    result = (x[0] - 2.0 * x[1] + 1.0);
                    return result;
                case 2:
                    result = (-x[0] * x[0]) / 4.0 -
                              (x[1] * x[1]) + 1.0);
                    return result;
                default:
                    ierr[0] = true;
                    return 0.0;
            }
        }
    }

    public void Gradient(double[] x, int iact, double[] result)
    {
        if (iact == 0)
        {
            result[0] = 2.0 * (x[0] - 2.0);
            result[1] = 2.0 * (x[1] - 1.0);
            return;
        }
    }
}
else
{
    switch (iact)
    {
    case 1:
        result[0] = 1.0;
        result[1] = -2.0;
        return;

    case 2:
        result[0] = -0.5 * x[0];
        result[1] = -2.0 * x[1];
        return;
    }
}

public static void Main(String[] args)
{
    int m = 2;
    int me = 1;
    int n = 2;
    MinConNLP minconnon = new MinConNLP(m, me, n);
    minconnon.SetGuess(new double[]{2.0, 2.0});
    double[] x = minconnon.Solve(new MinConNLPEx2());
    new PrintMatrix("x").Print(x);
}

Output

x

0 0.8228756553253
1 0.91143782776626

MinConNLP.IFunction Interface

public interface Imsl.Math.MinConNLP.IFunction

Public interface for the user supplied function to the MinConNLP object.
**Method**

**F**

abstract public double F(double[] x, int iact, bool[] ierr)

**Description**
Compute the value of the function at the given point.

**Parameters**
- **x** – An input double array, the point at which the objective function or constraint is to be evaluated.
- **iact** – An input int value indicating whether evaluation of the objective function is requested or evaluation of a constraint is requested. If iact is zero, then an objective function evaluation is requested. If iact is nonzero then the value of iact indicates the index of the constraint to evaluate. (1 indicates the first constraint, 2 indicates the second, etc.)
- **ierr** – An input/output bool array of length 1. On input ierr[0] is set to false. If an error or other undesirable condition occurs during evaluation, then ierr[0] should be set to true. Setting ierr[0] to true will result in the step size being reduced and the step being tried again. (If ierr[0] is set to true for xguess, then an error is issued.)

**Returns**
A double. If iact is zero, then the value of the objective function at x is returned. If iact is nonzero, then the computed constraint value at the point x is returned.

---

**MinConNLP.IGradient Interface**

class MinConNLP.IGradient : MinConNLP.IFunction

Public interface for the user supplied function to compute the gradient for MinConNLP object.

---

**Method**

**Gradient**

abstract public void Gradient(double[] x, int iact, double[] result)

**Description**
Computes the value of the gradient of the function at the given point.

**Parameters**
- **x** – An input double array, the point at which the gradient of the objective function or gradient of a constraint is to be evaluated.
iact – An input int value indicating whether evaluation of the objective function gradient is requested or evaluation of a constraint gradient is requested. If iact is zero, then an objective function gradient evaluation is requested. If iact is nonzero then the value of iact indicates the index of the constraint gradient to evaluate. (1 indicates the first constraint, 2 indicates the second, etc.)

result – A double array. If iact is zero, then the value of the objective function gradient at x is returned in result. If iact is nonzero, then the computed gradient of the requested constraint value at the point x is returned in result.

---

### NumericalDerivatives Class

```java
public class Imsl.Math.NumericalDerivatives
```

Compute the Jacobian matrix for a function \( f(y) \) with \( m \) components in \( n \) independent variables.

**NumericalDerivatives** uses divided finite differences to compute the Jacobian. This class is designed for use in numerical methods for solving nonlinear problems where a Jacobian is evaluated repeatedly at neighboring arguments. For example, this occurs in a Gauss-Newton method for solving non-linear least squares problems or a non-linear optimization method.

**NumericalDerivatives** is suited for applications where the Jacobian is a dense matrix. All cases \( m < n, m = n, \) or \( m > n \) are allowed. Both one-sided and central divided differences can be used.

The design allows for computation of derivatives in a variety of contexts. Note that a gradient should be considered as the special case with \( m = 1, n \geq 1 \). A derivative of a single function of one variable is the case \( m = 1, n = 1 \). Any non-linear solving routine that optionally requests a Jacobian or gradient can use **NumericalDerivatives**. This should be considered if there are special properties or scaling issues associated with \( f(y) \). Use the method `SetDifferencingMethods` to specify different differencing options for numerical differentiation. These can be combined with some analytic subexpressions or other known relationships.

The divided differences are computed using values of the independent variables at the initial point \( y_e = y \), and differenced points \( y_{e+j} = y + del \times e_j \). Here the \( e_j, j = 1, ..., n \), are the unit coordinate vectors. The value for each difference \( del \) depends on the variable \( j \), the differencing method, and the scaling for that variable. This difference is computed internally. See `SetPercentageFactor` for computational details. The evaluation of \( f(y_e) \) is normally done by the user-provided method `NumericalDerivatives.IFunction.F`, using the values \( y_e \). The index \( j \) and values \( y_e \) are arguments to `NumericalDerivatives.IFunction.F`.

The computational kernel of `EvaluateJ` performs the following steps:

1. evaluate the equations at the point \( y \) using `NumericalDerivatives.IFunction.F`.
2. compute the Jacobian.
3. compute the difference at \( y_e \).
By default, EvaluateJ uses NumericalDerivatives.IFunction.F in step 3. The user may choose to override the EvaluateF method to extend the capability of the class beyond the default.

There are six examples provided which illustrate various ways to use NumericalDerivatives. A discussion of the expected errors for these difference methods is found in *A First Course in Numerical Analysis*, Anthony Ralston, McGraw-Hill, NY, (1965).

### Constructor

#### NumericalDerivatives

```csharp
```

**Description**

Constructor for NumericalDerivatives.

**Parameter**

- `fcn` – An `IFunction` object which is a user-supplied function to evaluate the equations at the point `y`.

### Methods

#### EvaluateF

```csharp
virtual public double[] EvaluateF(int varIndex, double[] y)
```

**Description**

This method is provided by the user to compute the function values at the current independent variable values `y`.

**Parameters**

- `varIndex` – An `int` which indicates the index of the variable to perturb.
- `y` – A `double` array of length `n`, the point at which the function is to be evaluated.

**Returns**

A `double` array of length `m`. The equations evaluated at the point `y`.

**Remarks**

If the user does not override the `EvaluateF` method, then `NumericalDerivatives.IFunction.F` is used to compute the function values.

#### EvaluateJ

```csharp
public double[,] EvaluateJ(double[] y)
```
**Description**
Evaluates the Jacobian for a system of \( m \) equations in \( n \) variables.

**Parameter**
y – A double array of length \( n \), the point at which the Jacobian is to be evaluated.

**Returns**
A double matrix containing the Jacobian. Columns that are accumulated must have the additive term defined on entry or else be set to zero. Columns that are skipped can be defined either before or after the `EvaluateJ` method is invoked.

---

**GetPercentageFactor**
public double[] GetPercentageFactor()

**Description**
Returns the percentage factor for differencing.

**Returns**
A double array containing the percentage factor for differencing. See `SetPercentageFactor` for more detail.

---

**GetScalingFactors**
public double[] GetScalingFactors()

**Description**
Returns the scaling factors for the \( y \) values.

**Returns**
A double array containing the scaling factors.

---

**GetStats**
public int[] GetStats()

**Description**
Returns status information. This information might prove useful to the user wanting to gain better control over the differencing parameters. This information can often be ignored.

**Returns**
An int array containing the ten diagnostic values described in the following table. These values can be used to monitor the progress or expense of the Jacobian computation.
<table>
<thead>
<tr>
<th>index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the number of times a function evaluation was computed.</td>
</tr>
<tr>
<td>1</td>
<td>the number of columns in which three attempts were made to increase a percentage factor for differencing (i.e. a component in the factor array) but the computed del remained unacceptably small relative to y[j-1] or scale[j-1]. In such cases the percentage factor is set to 1.490116193847656e-8, which is the square root of machine precision.</td>
</tr>
<tr>
<td>2</td>
<td>the number of columns in which the computed del was zero to machine precision because y[j-1] or scale[j-1] was zero. In such cases del is set to 1.490116193847656e-8, which is the square root of machine precision.</td>
</tr>
<tr>
<td>3</td>
<td>the number of Jacobian columns which had to be recomputed because the largest difference formed in the column was close to zero relative to scale, where [ scale = \max \left(</td>
</tr>
<tr>
<td>4</td>
<td>the number of columns whose largest difference is close to zero relative to scale after the column has been recomputed.</td>
</tr>
<tr>
<td>5</td>
<td>the number of times scale information was not available for use in the roundoff and truncation error tests. This occurs when [ \min \left(</td>
</tr>
<tr>
<td>6</td>
<td>the number of times the increment for differencing (del) was computed and had to be increased because (scale[j-1]+del) - scale[j-1]) was too small relative to y[j-1] or scale[j-1].</td>
</tr>
<tr>
<td>7</td>
<td>the number of times a component of the factor array was reduced because changes in function values were large and excess truncation error was suspected. index = 8 gives the last column in which this occurred.</td>
</tr>
<tr>
<td>8</td>
<td>the index of the last column where the corresponding component of the factor array had to be reduced because excessive truncation error was suspected.</td>
</tr>
<tr>
<td>9</td>
<td>the index of the last column where the difference was small and the column had to be recomputed with an adjusted increment (see index = 3). The largest derivative in this column may be inaccurate due to excessive round-off error.</td>
</tr>
</tbody>
</table>

**SetDifferencingMethods**

```csharp
```

**Description**

Sets the methods used to compute the derivatives.
Parameter

options – A DifferenceMethod array of length n, containing the methods used to compute the derivatives. options[i] is the method to be used for the i-th variable. options[i] can be one of the values in the table which follows.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OneSided</td>
<td>Indicates one sided differences.</td>
</tr>
<tr>
<td>Central</td>
<td>Indicates central differences.</td>
</tr>
<tr>
<td>Accumulate</td>
<td>Indicates the accumulation of the result from whatever type of differences have been specified previously into initial values of the Jacobian.</td>
</tr>
<tr>
<td>Skip</td>
<td>Indicates a variable to be skipped.</td>
</tr>
</tbody>
</table>

Default: OneSided differences are used for each variable.

SetInitialF

public void SetInitialF(double[] valueF)

Description
Set the initial function values.

Parameter

valueF – A double array of length m containing the initial function values, y0.
Default: All values are 0.0.

Remarks
Use the values f(y0), where y0 is the initial value of the independent variables located in array y.

SetPercentageFactor

public void SetPercentageFactor(double[] factor)

Description
Sets the percentage factor for differencing.

Parameter

factor – A double array of length n containing the percentage factor for differencing. Except for initialization, the factor array should not be altered in the EvaluateF method. The elements of factor must be such that

\[1.8189894035458565 \times 10^{-12} \leq |factor[j - 1]| \leq 0.1\]

where 1.8189894035458565e-12 is machine precision to the three-fourths power.
Default: all elements of factor are set to \(1.4901161193847656e-8\), which is the square root of machine precision.
Remarks
For each divided difference for variable \( j \) the increment used is \( \delta \). The value of \( \delta \) is computed as follows: First define \( \sigma = \text{sign}([\text{scale}[j - 1]]) \). If the user has set the elements of array \( \text{scale} \) to non-default values, then define \( y_a = |\text{scale}[j - 1]| \). Otherwise, \( y_a = |y[j - 1]| \) and \( \sigma = 1 \). Finally, compute \( \delta = \sigma y_a \) factor\( [j - 1] \). By changing the sign of \( \text{scale}[j-1] \), the difference \( \delta \) can have any desired orientation, such as staying within bounds on variable \( j \). For central differences, a reduced factor is used for \( \delta \) that normally results in relative errors as small as machine precision to the 2/3 power.

**SetScalingFactors**

```csharp
public void SetScalingFactors(double[] scale)
```

**Description**
Sets the scaling factors for the \( y \) values. The user can also use \( \text{scale} \) to provide appropriate signs for the increments.

**Parameter**

- \( \text{scale} \) – A double array of length \( n \) containing the scaling factors.
  Default: All values are 1.0.

**Example 1: One-Sided Differences**
A simple use of \texttt{NumericalDerivatives} is shown. The gradient of the function \( f(y_1, y_2) = a \exp(by_1) + cy_1y_2^2 \) is required for values \( a = 2.5e6, b = 3.4, c = 4.5, y_1 = 2.1, y_2 = 3.2 \) The numerical gradient is compared to the analytic gradient, cast as a 1 by 2 Jacobian:

\[
\text{grad}(f) = \begin{bmatrix} ab \exp(by_1) + cy_2^2, & 2cy_1y_2 \end{bmatrix}
\]

This analytic gradient is expected to approximately agree with the numerical differentiation gradient. Relative agreement should be approximately the square root of machine precision. That is achieved here. Generally this is the most accuracy one can expect using one-sided divided differences.

```csharp
using System;
using Imsl.Math;

public class NumericalDerivativesEx1 : NumericalDerivatives.IFunction
{
    static int m = 1, n = 2;
    static double a, b, c;
    // This sets the function value used in forming one-sided
    // differences.
    public double[] F(int varIndex, double[] y)
    {
        double[] tmp = new double[m];
        tmp[0] = a * Math.Exp(b * y[0]) + c * y[0] * y[1] * y[1];
        return tmp;
    }

    public static void Main(String[] args)
    {
    ...
    
436 • NumericalDerivatives
```

IMSL C# Numerical Library
{ 
    double u;
    double[] y = new double[n];
    double[] scale = new double[n];
    double[,] actual = new double[m, n];
    double[] re = new double[2];

    // Define data and point of evaluation:
    a = 2.5e6;
    b = 3.4e0;
    c = 4.5e0;
    y[0] = 2.1e0;
    y[1] = 3.2e0;

    // Precision for measuring errors
    u = Math.Sqrt(2.220446049250313e-016);

    // Set scaling:
    scale[0] = 1.0e0;
    // Increase scale to account for large value of a.
    scale[1] = 8.0e3;

    // Compute true values of partials.
    actual[0, 0] = a * b * Math.Exp(b * y[0]) + c * y[1] * y[1];
    actual[0, 1] = 2 * c * y[0] * y[1];

    NumericalDerivatives derv =
        new NumericalDerivatives(new NumericalDerivativesEx1());
    derv.SetScalingFactors(scale);
    double[,] jacobian = derv.EvaluateJ(y);

    PrintMatrixFormat pmf = new PrintMatrixFormat();
    pmf.NumberFormat = "0.00";
    new PrintMatrix("Numerical gradient:").Print(pmf, jacobian);
    new PrintMatrix("Analytic gradient:").Print(pmf, actual);

    // Check the relative accuracy of one-sided differences.
    // They should be good to about half-precision.
    jacobian[0, 0] = (jacobian[0, 0] - actual[0, 0]) / actual[0, 0];
    jacobian[0, 1] = (jacobian[0, 1] - actual[0, 1]) / actual[0, 1];
    re[0] = jacobian[0, 0];
    re[1] = jacobian[0, 1];

    Console.Out.WriteLine("Relative accuracy:");
    Console.Out.WriteLine("df/dy_1  df/dy_2");
    Console.Out.WriteLine("{0:F2}u  {1:F2}u", re[0]/u, re[1]/u);
    Console.Out.WriteLine("({0:#.###e+00})  ({1:#.###e+00})", re[0], re[1]);
}

Output
Numerical gradient:
Analytic gradient:

```
0 1
Analytic gradient:
0 1
0 10722141696.00 60.48
```

Relative accuracy:
```
df/dy_1   df/dy_2
2.14u 0.00u
(3.195e-08) (-1.175e-16)
```

Example 2: Skipping A Gradient Component

This example uses the same data as in the One-Sided Differences example. Here we assume that the second component of the gradient is analytically known. Therefore only the first gradient component needs numerical approximation. The input values of array options specify that numerical differentiation with respect to \(y_2\) is skipped.

```
using System;
using Imsl.Math;

public class NumericalDerivativesEx2 : NumericalDerivatives.IJacobian
{
    static int m = 1, n = 2;
    static double a, b, c;

    public double[] F(int varIndex, double[] y)
    {
        double[] tmp = new double[m];
        tmp[0] = a * Math.Exp(b * y[0]) + c * y[0] * y[1] * y[1];
        return tmp;
    }

    public double[,] Jacobian(double[] y)
    {
        double[,] tmp = new double[m, n];

        // The second component partial is skipped,
        // since it is known analytically
        tmp[0, 1] = 2.0e0 * c * y[0] * y[1];
        return tmp;
    }

    public static void Main(String[] args)
    {
        NumericalDerivatives.DifferencingMethod[] options =
            new NumericalDerivatives.DifferencingMethod[n];
        double u;
        double[] y = new double[n];
        double[] valueF = new double[m];
```
double[] scale = new double[n];
double[,] actual = new double[m, n];
double[] re = new double[2];

// Define data and point of evaluation:
a = 2.5e6;
b = 3.4e0;
c = 4.5e0;
y[0] = 2.1e0;
y[1] = 3.2e0;

// Precision, for measuring errors
u = Math.Sqrt(2.220446049250313e-016);

// Set scaling:
scale[0] = 1.0e0;
// Increase scale to account for large value of a.
scale[1] = 8.0e3;

// Compute true values of partials.
actual[0, 0] = a * b * Math.Exp(b * y[0]) + c * y[1] * y[1];
actual[0, 1] = 2 * c * y[0] * y[1];

options[0] = NumericalDerivatives.DifferencingMethod.OneSided;

valueF[0] = a * Math.Exp(b * y[0]) + c * y[0] * y[1] * y[1];
NumericalDerivatives deriv =
    new NumericalDerivatives(new NumericalDerivativesEx2());
deriv.SetDifferencingMethods(options);
deriv.SetScalingFactors(scale);
deriv.SetInitialF(valueF);
double[,] jacobian = deriv.EvaluateJ(y);

PrintMatrixFormat pmf = new PrintMatrixFormat();
new PrintMatrix("Numerical gradient:").Print(pmf, jacobian);
new PrintMatrix("Analytic gradient:").Print(pmf, actual);

jacobian[0, 0] = (jacobian[0, 0] - actual[0, 0]) / actual[0, 0];
jacobian[0, 1] = (jacobian[0, 1] - actual[0, 1]) / actual[0, 1];
re[0] = jacobian[0, 0];
re[1] = jacobian[0, 1];

Console.Out.WriteLine("Relative accuracy:");
Console.Out.WriteLine(" df/dy_1   df/dy_2");
Console.Out.WriteLine(" {0:F2}u   {1:F2}u", 
    re[0]/u, re[1]/u);
Console.Out.WriteLine(" {{0:#.###e+00}}  {{1:#.###e+00}}", 
    re[0], re[1]);
}
Example 3: Accumulation Of A Component

This example uses the same data as in the One-Sided Differences example. An alternate examination of the function \( f(y_1, y_2) = a \exp(by_1) + cy_1^2 \) shows that the first term on the right-hand side need be evaluated just when computing the first partial. The additive term \( cy_1^2 \) occurs when computing the partial with respect to \( y_1 \). Also the first term does not depend on the second variable. Thus the first term can be left out of the function evaluation when computing the partial with respect to \( y_2 \), potentially avoiding cancellation errors. The input values of array options allow NumericalDerivatives to use these facts and obtain greater accuracy using a minimum number of computations of the exponential function.

```csharp
using System;
using Imsl.Math;

public class NumericalDerivativesEx3 : NumericalDerivatives.IJacobian
{
    static int m = 1, n = 2;
    static double a, b, c;

    public double[] F(int varIndex, double[] y)
    {
        double[] tmp = new double[m];
        if (varIndex != 2)
        {
            tmp[0] = a * Math.Exp(b * y[0]);
        }
        else
        {
            // This is the function value for the partial wrt y_2.
            tmp[0] = c * y[0] * y[1] * y[1];
        }
        return tmp;
    }

    public double[,] Jacobian(double[] y)
    {
        // Code...
    }
}
double[,] tmp = new double[m, n];

// Start with part of the derivative that is known.
tmp[0, 0] = c * y[1] * y[1];

return tmp;
}

public static void Main(String[] args)
{
    NumericalDerivatives.DifferencingMethod[] options =
        new NumericalDerivatives.DifferencingMethod[n];
double u;
double[] y = new double[n];
double[] valueF = new double[m];
double[] scale = new double[n];
double[,] actual = new double[m, n];
double[] re = new double[2];

    // Define data and point of evaluation:
a = 2.5e6;
b = 3.4e0;
c = 4.5e0;
y[0] = 2.1e0;
y[1] = 3.2e0;

    // Precision, for measuring errors
    u = Math.Sqrt(2.220446049250313e-016);

    // Set scaling:
scale[0] = 1.0e0;
    // Increase scale to account for large value of a.
scale[1] = 8.0e3;

    // Compute true values of partials.
    actual[0, 0] = a * b * Math.Exp(b * y[0]) + c * y[1] * y[1];
    actual[0, 1] = 2 * c * y[0] * y[1];


    valueF[0] = a * Math.Exp(b * y[0]);
    scale[1] = 1.0e0;

    NumericalDerivatives deriv =
        new NumericalDerivatives(new NumericalDerivativesEx3());
    deriv.SetDifferencingMethods(options);
    deriv.SetScalingFactors(scale);
    deriv.SetInitialF(valueF);
    double[,] jacobian = deriv.EvaluateJ(y);

    PrintMatrixFormat pmf = new PrintMatrixFormat();
    pmf.NumberFormat = "0.00";
    new PrintMatrix("Numerical gradient:").Print(pmf, jacobian);
    new PrintMatrix("Analytic gradient:").Print(pmf, actual);
jacobian[0, 0] = (jacobian[0, 0] - actual[0, 0]) / actual[0, 0];
jacobian[0, 1] = (jacobian[0, 1] - actual[0, 1]) / actual[0, 1];
re[0] = jacobian[0, 0];
re[1] = jacobian[0, 1];

Console.Out.WriteLine("Relative accuracy:");
Console.Out.WriteLine("df/dy_1   df/dy_2");
Console.Out.WriteLine(" {0:F2}u  {1:F2}u",
re[0]/u, re[1]/u);
Console.Out.WriteLine("({0:#.###e+00})  ({1:#.###e+00})",
re[0], re[1]);

Output

Numerical gradient:
0 1
0 10722141710.08 60.48

Analytic gradient:
0 1
0 10722141353.42 60.48

Relative accuracy:
df/dy_1   df/dy_2
2.23u   -0.51u
(3.326e-08)  (-7.569e-09)

Example 4: Central Differences

This example uses the same data as in the One-Sided Differences example. Agreement should be
approximately the two-thirds power of machine precision. That agreement is achieved here. Generally
this is the most accuracy one can expect using central divided differences. Note that using central
differences requires essentially twice the number of evaluations of the function compared with obtaining
one-sided differences. This can be a significant issue for functions that are expensive to evaluate. This
example shows how to override EvaluateF.

using System;
using Imsl.Math;

public class NumericalDerivativesEx4 : NumericalDerivatives
{
    static int m = 1, n = 2;
    static double a, b, c, v = 0.0;

    class NumericalDerivativesFcn : NumericalDerivatives.IFunction
    {
        public double[] F(int varIndex, double[] y)
        {

        }
return new double[m];
}

public NumericalDerivativesEx4(NumericalDerivatives.IFunction fcn):
    base(fcn)
{
}

// Override EvaluateF.
public override double[] EvaluateF(int varIndex, double[] y)
{
    double[] valueF = new double[m];
    valueF[0] = a * Math.Exp(b * y[0]) + c * y[0] * y[1] * y[1];
    return valueF;
}

public static void Main(String[] args)
{
    NumericalDerivatives.DifferencingMethod[] options =
        new NumericalDerivatives.DifferencingMethod[n];
    double u;
    double[] y = new double[n];
    double[] scale = new double[n];
    double[,] actual = new double[m, n];
    double[] re = new double[2];
    // Define data and point of evaluation:
    a = 2.5e6;
    b = 3.4e0;
    c = 4.5e0;
    y[0] = 2.1e0;
    y[1] = 3.2e0;
    // Machine precision, for measuring errors
    u = 2.220446049250313e-016;
    v = Math.Pow(3.0e0 * u, 2.0e0 / 3.0e0);
    // Set scaling:
    scale[0] = 1.0e0;
    // Increase scale to account for large value of a.
    scale[1] = 8.0e3;
    // Compute true values of partials.
    actual[0, 0] = a * b * Math.Exp(b * y[0]) + c * y[1] * y[1];
    actual[0, 1] = 2 * c * y[0] * y[1];
    options[0] = NumericalDerivatives.DifferencingMethod.Central;
    // Set the increment used at the default value.
    scale[1] = 8.0e3;
NumericalDerivativesEx4 derv =
    new NumericalDerivativesEx4(new NumericalDerivativesFcn());
derv.SetDifferencingMethods(options);
derv.SetScalingFactors(scale);
double[,] jacobian = derv.EvaluateJ(y);

PrintMatrixFormat pmf = new PrintMatrixFormat();
    pmf.NumberFormat = "0.00";
new PrintMatrix("Numerical gradient:").Print(pmf, jacobian);
new PrintMatrix("Analytic gradient:").Print(pmf, actual);

    // Since the function is never evaluated at the
    // initial point, hold back until the request is made.

    // Check the relative accuracy of central differences.
    // They should be good to about two thirds-precision.
    jacobian[0, 0] = (jacobian[0, 0] - actual[0, 0]) / actual[0, 0];
    jacobian[0, 1] = (jacobian[0, 1] - actual[0, 1]) / actual[0, 1];
    re[0] = jacobian[0, 0];
    re[1] = jacobian[0, 1];

    Console.Out.WriteLine("Relative accuracy:");
    Console.Out.WriteLine("df/dy_1 df/dy_2");
    Console.Out.WriteLine(" {0:F2}v {1:F2}v", re[0]/v, re[1]/v);
    Console.Out.WriteLine("({0:#.###e+00}) ({1:#.###e+00})", re[0], re[1]);

//Example 5: Hessian Approximation

This example uses the same data as in the One-Sided Differences example. In this example numerical
differentiation is used to approximate the Hessian matrix of \( f(y_1, y_2) \). This symmetric Hessian matrix is

\[
H_f = \begin{bmatrix}
\frac{\partial^2 f}{\partial y_1^2} & \frac{\partial^2 f}{\partial y_1 \partial y_2} \\
\frac{\partial^2 f}{\partial y_1 \partial y_2} & \frac{\partial^2 f}{\partial y_2^2}
\end{bmatrix}
\]
Our method is based on casting the matrix $Hf$ as the 2 by 2 Jacobian matrix of the gradient function. Each inner evaluation of the gradient function is itself computed using NumericalDerivatives. Central differences are used for both the inner and outer numerical differentiation. Because of the inherent error in both processes, the expected accuracy is about the $4/9 = (2/3)^2$ power of machine precision. Note that the approximation obtained is not symmetric. However, the difference between the off-diagonal elements provides an error estimate of that term.

```csharp
using System;
using Imsl.Math;

public class NumericalDerivativesEx5 : NumericalDerivatives
{
    static int m = 1, n = 2;
    static double a, b, c, v = 0.0;

    class NumericalDerivativesFcn : NumericalDerivatives.IFunction
    {
        private int num;

        public NumericalDerivativesFcn(int num)
        {
            this.num = num;
        }

        public double[] F(int varIndex, double[] y)
        {
            return new double[num];
        }
    }

    class InnerNumericalDerivatives : NumericalDerivatives
    {
        public InnerNumericalDerivatives(NumericalDerivatives.IFunction fcn) : base(fcn)
        {
        }

        // Override EvaluateF.
        public override double[] EvaluateF(int varIndex, double[] y)
        {
            double[] valueF = new double[m];

            valueF[0] = a * Math.Exp(b * y[0]) + c * y[0] * y[1] * y[1];
            return valueF;
        }
    }

    public NumericalDerivativesEx5(NumericalDerivatives.IFunction fcn) : base(fcn)
    {
    }
}
```
// Override EvaluateF.
public override double[,] EvaluateF(int varIndex, double[] y)
{
    NumericalDerivatives.DifferencingMethod[] iopt =
        new NumericalDerivatives.DifferencingMethod[n];
    double[] valueF = new double[m];
    double[] scale = new double[n];
    double[] fac = new double[n];

    // This is the analytic gradient. for comparison only.
    // stateh(1)=a*b*exp(b*y(1))+c*y(2)**2
    // stateh(2)=2*c*y(1)*y(2)
    //
    // Each request for a gradient evaluation uses the
    // functionality of numerical evaluation, but with
    // the same numerical code.
    iopt[0] = NumericalDerivatives.DifferencingMethod.Central;

    // Set the increment used at the default value.
    //
    // Set defaults for increments and scaling:
    fac[0] = 1.4901161193847656E-8;
    fac[1] = 1.4901161193847656E-8;

    // Change scale to account for large value of a.
    switch (varIndex)
    {  
        case 1:
            scale[0] = 1.0e0;
            scale[1] = 8.0e8;
            break;
        case 2:
            scale[0] = 1.0e4;
            scale[1] = 8.0e8;
            break;
    }

    InnerNumericalDerivatives derv =
        new InnerNumericalDerivatives(new NumericalDerivativesFcn(m));
    derv.SetPercentageFactor(fac);
    derv.SetDifferencingMethods(iopt);
    derv.SetScalingFactors(scale);
    derv.SetInitialF(valueF);
    double[,] fjac = derv.EvaluateJ(y);

    // Since the function is never evaluated at the
    // initial point, hold back until the request is made.

    // Copy gradient value into array expected by
    // outer loop computing the Hessian matrix.
    double[] tmp = new double[n];
    for (int i = 0; i < n; i++)
    {
        tmp[i] = fjac[0, i];
    }
}
public static void Main(String[] args) {
    NumericalDerivatives.DifferencingMethod[] iopth =
            new NumericalDerivatives.DifferencingMethod[n];
    double u;
    double[] fach = new double[n];
    double[] stateh = new double[n];
    double[] scaleh = new double[n];
    double[,] actual = new double[n, n];
    double[] y = new double[n];

    // Define data and point of evaluation:
    a = 2.5e6;
    b = 3.4e0;
    c = 4.5e0;
    y[0] = 2.1e0;
    y[1] = 3.2e0;

    // Machine precision, for measuring errors
    u = 2.220446049250313e-016;

    // Compute expected relative error using two applications
    // of central differences.
    v = Math.Pow(3.0e0 * u, 2.0e0 / 3.0e0);
    v = Math.Pow(3 * v, 2.0 / 3.0);

    // Set increments and scaling:
    fach[0] = 1.4901161193847656E-8;
    fach[1] = 1.4901161193847656E-8;
    iopth[0] = NumericalDerivatives.DifferencingMethod.Central;

    // Compute true values of partials.
    actual[0, 0] = a * b * b * Math.Exp(b * y[0]);
    actual[1, 0] = 2 * c * y[1];
    actual[0, 1] = 2 * c * y[1];
    actual[1, 1] = 2 * c * y[0];

    // Set the increment used at the default value.
    scaleh[0] = 1;
    scaleh[1] = 8.0e5;

    NumericalDerivativesEx5 derv2 =
            new NumericalDerivativesEx5(new NumericalDerivativesFcn(n));
    derv2.SetPercentageFactor(fach);
    derv2.SetDifferencingMethods(iopth);
    derv2.SetScalingFactors(scaleh);
    derv2.SetInitialF(stateh);
    double[,] h = derv2.EvaluateJ(y);

    PrintMatrixFormat pmf = new PrintMatrixFormat();
    pmf.NumberFormat = "0.00";
new PrintMatrix("Numerical Hessian:").Print(pmf, h);
new PrintMatrix("Analytic Hessian:").Print(pmf, actual);

// Since the function is never evaluated at the
// initial point, hold back until the request is made.

// Subtract the actual hessian matrix values and check.
h[0, 0] = (h[0, 0] - actual[0, 0]) / h[0, 0] / v;
h[1, 0] = (h[1, 0] - actual[1, 0]) / h[1, 0] / v;
h[0, 1] = (h[0, 1] - actual[0, 1]) / h[0, 1] / v;
h[1, 1] = (h[1, 1] - actual[1, 1]) / h[1, 1] / v;

pmf.NumberFormat = "0.000";
new PrintMatrix("Hessian Matrix, Expected Normalized " +
    "Relative Error, |all entries|").Print(pmf, h);

Output

Numerical Hessian:
0 1
0 36455292905.82 28.80
1 28.80 18.90

Analytic Hessian:
0 1
0 36455280444.94 28.80
1 28.80 18.90

Hessian Matrix, Expected Normalized Relative Error, |all entries|
0 1
0 0.914 0.037
1 0.036 0.000

Example 6: Usage With Class MinUnconMultiVar

The minimum of $100(x_2-x_1^2)^2 + (1-x_1)^2$ is found using MinUnconMultiVar. NumericalDerivatives is used to compute the numerical gradients.

using System;
using Imsl.Math;

public class NumericalDerivativesEx6 : MinUnconMultiVar.IGradient
{
    static int m = 1, n = 2;

    class NumericalDerivativesFcn : NumericalDerivatives.IFunction
    {
        public double[] F(int varIndex, double[] y)
        {
            double[] tmp = new double[m];
        }
```java
    tmp[0] = NumericalDerivativesEx6.FcnEvaluation(y);
    return tmp;
}
}
static public double FcnEvaluation(double[] x)
{
    return 100.0 * ((x[1] - x[0] * x[0]) * (x[1] - x[0] * x[0])) +
               (1.0 - x[0]) * (1.0 - x[0]);
}
public double F(double[] x)
{
    return FcnEvaluation(x);
}
public void Gradient(double[] x, double[] gp)
{
    NumericalDerivatives nderv =
        new NumericalDerivatives(new NumericalDerivativesFcn());
    double[,] jacobian = nderv.EvaluateJ(x);
    gp[0] = jacobian[0, 0];
    gp[1] = jacobian[0, 1];
}

public static void Main(String[] args)
{
    MinUnconMultiVar solver = new MinUnconMultiVar(n);
    solver.SetGuess(new double[]{ -1.2, 1.0 });
    double[] x = solver.ComputeMin(new NumericalDerivativesEx6());
    Console.Out.WriteLine
        ("Minimum point is (" + x[0] + ", " + x[1] + ")");
}
}

Output
Minimum point is (0.999998611858024, 0.999997274648157)

NumericalDerivatives.IFunction Interface

Public interface function.

Optimization NumericalDerivatives.IFunction • 449
Method

F
abstract public double[] F(int varIndex, double[] y)

Description
Returns the equations evaluated at the point y. If the user does not override the EvaluateF method, then F is also used to compute the function values at the current independent variable values y.

Parameters

varIndex – An int indicating the index of the variable to perturb. varIndex = 1 indicates variable 1 in y[0].
y – A double array of length n, the point at which the Jacobian is to be evaluated.

Returns
A double array of length m. The equations evaluated at the point y.

NumericalDerivatives.IJacobian Interface

Public interface for the user-supplied function to compute the Jacobian.

Method

Jacobian
abstract public double[,] Jacobian(double[] y)

Description
User-supplied function to compute the Jacobian.

Parameter

y – A double array of length n, the point at which the Jacobian is to be evaluated.

Returns
A double m by n matrix containing the Jacobian. Columns that are accumulated must have the analytic part defined on entry or else be set to zero. Columns that are skipped can be defined either before or after the EvaluateJ method is invoked.
**NumericalDerivatives.DifferencingMethod Enumeration**


Specifies the differencing method.

**Fields**

**Accumulate**


**Description**

Indicates the accumulation of the result from whatever type of differences have been specified previously into initial values of the Jacobian.

**Central**


**Description**

Indicates central differences.

**OneSided**


**Description**

Indicates one sided differences.

**Skip**


**Description**

Indicates a variable to be skipped.
Chapter 9: Special Functions

Types

class Sfun ......................................................... 453
class Bessel ....................................................... 469

Sfun Class

public class Imsl.Math.Sfun
Collection of special functions.

Fields

EpsilonLarge
public double EpsilonLarge

Description
The largest relative spacing for doubles.

EpsilonSmall
public double EpsilonSmall

Description
The smallest relative spacing for doubles.
Methods

Asinh
static public double Asinh(double x)

Description
Returns the hyperbolic arc sine of a double.

Parameter
x – A double value for which the hyperbolic arc sine is desired.

Returns
A double specifying the hyperbolic arc sine value.

Beta
static public double Beta(double a, double b)

Description
Returns the value of the Beta function.

Parameters
a – A double value.
b – A double value.

Returns
A double value specifying the Beta function.

Remarks
The Beta function is defined to be

\[ \beta(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} = \int_0^1 t^{a-1}(1-t)^{b-1} dt \]

See Gamma for the definition of \( \Gamma(x) \).
The method Beta requires that both arguments be positive.

BetaIncomplete
static public double BetaIncomplete(double x, double p, double q)

Description
Returns the incomplete Beta function ratio.

Parameters
x – A double value specifying the upper limit of integration It must be in the interval [0,1] inclusive.
p – A double value specifying the first Beta parameter. It must be positive.
q – A double value specifying the second Beta parameter. It must be positive.
Returns
A double value specifying the incomplete Beta function ratio.

Remarks
The incomplete beta function is defined to be

\[ I_x(p, q) = \frac{\beta_x(p, q)}{\beta(p, q)} = \frac{1}{\beta(p, q)} \int_0^x t^{p-1} (1-t)^{q-1} dt \]
for \( 0 \leq x \leq 1, p > 0, q > 0 \)

See Beta for the definition of \( \beta(p, q) \).

The parameters \( p \) and \( q \) must both be greater than zero. The argument \( x \) must lie in the range 0 to 1. The incomplete beta function can underflow for sufficiently small \( x \) and large \( p \); however, this underflow is not reported as an error. Instead, the value zero is returned as the function value.

The method BetaIncomplete is based on the work of Bosten and Battiste (1974).

Cot

static public double Cot(double x)

Description
Returns the cotangent of a double.

Parameter
x – A double value

Returns
A double value specifying the cotangent of \( x \). If \( x \) is NaN, the result is NaN.

Erf

static public double Erf(double x)

Description
Returns the error function of a double.

Parameter
x – A double value.

Returns
A double value specifying the error function of \( x \).

Remarks
The error function method, \( \text{Erf}(x) \), is defined to be

\[ \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \]

All values of \( x \) are legal.
**Erfc**

```csharp
static public double Erfc(double x)
```

**Description**
Returns the complementary error function of a double.

**Parameter**
- `x` – A double value.

**Returns**
A double value specifying the complementary error function of `x`.

**Remarks**
The complementary error function method, Erfc(`x`), is defined to be
\[ \text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt \]

The argument \( x \) must not be so large that the result underflows. Approximately, \( x \) should be less than

\[ [-\ln(\sqrt{s})]^{1/2} \]

where \( s = \text{Double.Epsilon} \) is the smallest representable positive floating-point number.

---

**Erfce**

static public double Erfce(double x)

---

Special Functions

Sfun • 457
**Description**

Returns the exponentially scaled complementary error function.

**Parameter**

- $x$ – A double value for which the function value is desired.

**Returns**

A double value specifying the exponentially scaled complementary error function of $x$.

**Remarks**

The exponentially scaled complementary error function is defined as

$$e^{x^2} \text{erfc}(x)$$

where $\text{erfc}(x)$ is the complementary error function. See Erfc (p. 456) for its definition. To prevent the answer from underflowing, $x$ must be greater than

$$x_{\text{min}} \simeq -\sqrt{\ln(b/2)} = -26.618735713751487$$

where $b = \text{Double.MaxValue}$ is the largest representable double precision number.

**ErfcInverse**

static public double ErfcInverse(double x)

**Description**

Returns the inverse of the complementary error function.

**Parameter**

- $x$ – A double value, $0 \leq x \leq 2$.

**Returns**

A double value specifying the inverse of the error function of $x$.

**Remarks**

The ErfcInverse($x$) method computes the inverse of the complementary error function $\text{erfc} x$, defined in Erfc. ErfcInverse($x$) is defined for $0 < x < 2$. If $x_{\text{max}} < x < 2$, then the answer will be less accurate than half precision. Very approximately,

$$x_{\text{max}} \approx 2 - \sqrt{\varepsilon/(4\pi)}$$

where $\varepsilon = \text{machine precision}$ (approximately $1.11\text{e}-16$).
**ErfInverse**

```java
static public double ErfInverse(double x)
```

**Description**

Returns the inverse of the error function.

**Parameter**

- `x` – A double value.

**Returns**

A double value specifying the inverse of the error function of `x`.

**Remarks**

ErfInverse(X) method computes the inverse of the error function erf `x`, defined in Erf.
The method \texttt{ErfInverse(X)} is defined for $x_{\text{max}} < |x| < 1$, then the answer will be less accurate than half precision. Very approximately,

$$x_{\text{max}} \approx 1 - \sqrt{\varepsilon/(4\pi)}$$

where $\varepsilon$ is the machine precision (approximately 1.11e-16).

Fact

\texttt{static public double Fact(int n)}

Description

Returns the factorial of an integer.
**Parameter**

- \( n \) – An int value.

**Returns**

A double value specifying the factorial of \( n \), \( n! \). If \( x \) is negative, the result is NaN.

### Gamma

static public double Gamma(double x)

**Description**

Returns the Gamma function of a double.

**Parameter**

- \( x \) – A double value.

**Returns**

A double value specifying the Gamma function of \( x \). If \( x \) is a negative integer, the result is NaN.

**Remarks**

The Gamma function, \( \Gamma(x) \), is defined to be

\[
\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \quad \text{for} \ x > 0
\]

For \( x < 0 \), the above definition is extended by analytic continuation.

The Gamma function is not defined for integers less than or equal to zero. Also, the argument \( x \) must be greater than \(-170.56\) so that \( \Gamma(x) \) does not underflow, and \( x \) must be less than \( 171.64 \) so that \( \Gamma(x) \) does not overflow. The underflow limit occurs first for arguments that are close to large negative half integers. Even though other arguments away from these half integers may yield machine-representable values of \( \Gamma(x) \), such arguments are considered illegal. Users who need such values should use the Log Gamma. Finally, the argument should not be so close to a negative integer that the result is less accurate than half precision.
**GammaIncomplete**

static public double GammaIncomplete(double a, double x)

**Description**

Evaluates the incomplete gamma function.

**Parameters**

- **a** – A double value representing the integrand exponent parameter of the incomplete gamma function. If a is less than zero a `Double.NaN` is returned. equal to zero.
- **x** – A double value specifying the point at which the incomplete gamma function is to be evaluated. If x is less than zero or equal to zero, `Double.NaN` or 0.0 respectively is returned. nonnegative.

**Returns**

A double value specifying the incomplete gamma function.
Remarks
The lower limit of integration of the incomplete gamma function, \( \gamma(a, x) \), is defined to be
\[
\gamma(a, x) = \int_0^x t^{a-1} e^{-t} dt \quad \text{for} \quad x \geq 0 \text{ and } a > 0
\]
Although \( \gamma(a, x) \) is well defined for \( x > -\infty \), this algorithm does not calculate \( \gamma(a, x) \) for negative \( x \). For large \( a \) and sufficiently large \( x \), \( \gamma(a, x) \) may overflow. \( \gamma(a, x) \) is bounded by \( \Gamma(a) \), and users may find this bound a useful guide in determining legal values for \( a \).
Note that the upper limit of integration of the incomplete gamma, \( \Gamma(a, x) \), is defined to be
\[
\Gamma(a, x) = \int_x^\infty t^{a-1} e^{-t} dt
\]
Therefore, by definition, the two incomplete gamma function forms satisfy the relationship
\[
\Gamma(a, x) + \gamma(a, x) = \Gamma(a)
\]

**Log10**

static public double Log10(double x)

Description
Returns the common (base 10) logarithm of a double.

Parameter
\( x \) – A double value.

Returns
A double value specifying the common logarithm of \( x \).

**Log1p**

static public double Log1p(double x)

Description
Returns \( \log(1+x) \), the logarithm of \( (x + 1) \).

Parameter
\( x \) – A double value representing the argument.

Returns
A double value representing \( \log(1+x) \).
Remarks
Specifically:
\[ \log(1 + x) \] returns \( \pm 0 \).
\[ \log(-1) \] returns \(-\infty\).
\[ \log(x) \] returns NaN, if \( x < -1 \).
\[ \log(\pm \infty) \] returns \( \pm \infty \).

\[
\text{LogBeta}
\]
static public double LogBeta(double a, double b)

Description
Returns the logarithm of the Beta function.

Parameters
\begin{itemize}
  \item \texttt{a} – A double value.
  \item \texttt{b} – A double value.
\end{itemize}

Returns
A double value specifying the natural logarithm of the Beta function.

Remarks
Method \texttt{LogBeta} computes \[ \log(\beta(a, b)) = \log(\beta(b, a)) \]. See \texttt{Beta} for the definition of \( \beta(a, b) \).
\texttt{LogBeta} is defined for \( a > 0 \) and \( b > 0 \). It returns accurate results even when \( a \) or \( b \) is very small. It can overflow for very large arguments; this error condition is not detected except by the computer hardware.

\[
\text{LogGamma}
\]
static public double LogGamma(double x)

Description
Returns the logarithm of the Gamma function of the absolute value of a double.

Parameter
\begin{itemize}
  \item \texttt{x} – A double value.
\end{itemize}

Returns
A double value specifying the natural logarithm of the Gamma function of \( |x| \). If \( x \) is a negative integer, the result is NaN.

Remarks
Method \texttt{LogGamma} computes \[ \log(|\Gamma(x)|) \]. See \texttt{Gamma} for the definition of \( \Gamma(x) \).
The Gamma function is not defined for integers less than or equal to zero. Also, \( |x| \) must not be so large that the result overflows. Neither should \( x \) be so close to a negative integer that the accuracy is worse than half precision.
Poch

static public double Poch(double a, double x)

Description
Returns a generalization of Pochhammer’s symbol.

Parameters
a – A double value specifying the first argument.
x – A double value specifying the second, differential argument.

Returns
A double value specifying the generalized Pochhammer symbol, Gamma(a+x)/Gamma(a).

Remarks
Method Poch evaluates Pochhammer’s symbol \((a)_n = (a)(a-1)\ldots(a-n+1)\) for \(n\) a nonnegative integer. Pochhammer’s generalized symbol is defined to be
\[(a)_x = \frac{\Gamma(a + x)}{\Gamma(a)}\]

See Gamma for the definition of \(\Gamma(x)\).

Note that a straightforward evaluation of Pochhammer’s generalized symbol with either Gamma or Log Gamma functions can be especially unreliable when \(a\) is large or \(x\) is small.

Substantial loss can occur if \(a + x\) or \(a\) are close to a negative integer unless \(|x|\) is sufficiently small. To insure that the result does not overflow or underflow, one can keep the arguments \(a\) and \(a + x\) well within the range dictated by the Gamma function method Gamma or one can keep \(|x|\) small whenever \(a\) is large. Poch also works for a variety of arguments outside these rough limits, but any more general limits that are also useful are difficult to specify.

**Psi**

```csharp
static public double Psi(double x)
```

**Description**

Returns the derivative of the log gamma function, also called the digamma function.

**Parameter**

\(x\) – A double value, the point at which the digamma function is to be evaluated.

**Returns**

A double value specifying the logarithmic derivative of the gamma function of \(x\). If \(x\) is a zero or a negative integer, the result is NaN. If \(x\) is too close to a negative integer the accuracy of the result will be less than half precision.

**Remarks**

The psi function is defined to be

\[\psi(x) = \frac{d}{dx} \ln \Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)}\]

See Gamma (p. 461) for the definition of \(\Gamma(x)\).

The argument \(x\) must not be exactly zero or a negative integer, or \(\psi(x)\) is undefined. Also, \(x\) must not be too close to a negative integer such that the accuracy of the result is less than half precision.

**Psi1**

```csharp
static public double Psi1(double x)
```

**Description**

Returns the \(\psi_1\) function, also known as the trigamma function.

**Parameter**

\(x\) – A double value, the point at which the trigamma function is to be evaluated.

**Returns**

A double value specifying the trigamma function of \(x\). If \(x\) is a negative integer or zero, the result is NaN.
Remarks
The trigamma function, $\psi_1(x)$, is defined to be

$$\psi_1(x) = \frac{d^2}{dx^2} \ln \Gamma(x)$$

The trigamma function is not defined for integers less than or equal to zero.

R9lgmc
static public double R9lgmc(double x)

Description
Returns the Log Gamma correction term for argument values greater than or equal to 10.0.

Parameter
x – A double value.

Returns
A double value specifying the Log Gamma correction term.

Sign
static public double Sign(double x, double y)

Description
Returns the value of x with the sign of y.

Parameters
x – A double value.
    y – A double value.

Returns
A double value specifying the absolute value of x and the sign of y.

Example: The Special Functions
Various special functions are exercised. Their use in this example typifies the manner in which other special functions in the Sfun class would be used.

using System;
using Imsl.Math;

class SfunEx1
{
    public static void Main(String[] args)
    {
        double result;
        // Log base 10 of x
double x = 100.0;
result = Sfun.Log10(x);
Console.Out.WriteLine("The log base 10 of 100. is " + result);

// Factorial of 10
int n = 10;
result = Sfun.Fact(n);
Console.Out.WriteLine("10 factorial is " + result);

// Gamma of 5.0
double x1 = 5.0;
result = Sfun.Gamma(x1);
Console.Out.WriteLine("The Gamma function at 5.0 is " + result);

// LogGamma of 1.85
double x2 = 1.85;
result = Sfun.LogGamma(x2);
Console.Out.WriteLine("The logarithm of the absolute value of the Gamma function at 1.85 is " + result);

// Beta of (2.2, 3.7)
double a = 2.2;
double b = 3.7;
result = Sfun.Beta(a, b);
Console.Out.WriteLine("Beta(2.2, 3.7) is " + result);

// LogBeta of (2.2, 3.7)
double a1 = 2.2;
double b1 = 3.7;
result = Sfun.LogBeta(a1, b1);
Console.Out.WriteLine("logBeta(2.2, 3.7) is " + result + "\n");

Output
The log base 10 of 100. is 2
10 factorial is 3628800
The Gamma function at 5.0 is 24
The logarithm of the absolute value of the Gamma function at 1.85 is -0.0559238130196572
Beta(2.2, 3.7) is 0.0453759834847081
logBeta(2.2, 3.7) is -3.09277231203789
Bessel Class

public class Imsl.Math.Bessel
Collection of Bessel functions.

Methods

I

static public double[] I(double x, int n)

Description
Evaluates a sequence of modified Bessel functions of the first kind with integer order and real argument.

Parameters
- x – A double representing the argument of the Bessel functions to be evaluated.
- n – The int order of the last element in the sequence.

Returns
A double array of length n+1 containing the values of the function through the series.

Remarks
Bessel.I[i] contains the value of the Bessel function of order i. The Bessel function $I_n(x)$ is defined to be

$$I_n(x) = \frac{1}{\pi} \int_0^\pi e^x \cos \theta \cos(n\theta) \, d\theta$$

The input x must satisfy $|x| \leq \log(b)$ where b is the largest representable floating-point number. The algorithm is based on a code due to Sookne (1973b), which uses backward recursion.

I

static public double[] I(double xnu, double x, int n)

Description
Evaluates a sequence of modified Bessel functions of the first kind with real order and real argument.

Parameters
- xnu – A double representing the lowest order desired. xnu must be at least zero and less than 1.
- x – A double representing the argument of the Bessel functions to be evaluated.
- n – The int order of the last element in the sequence.

Returns
A double array of length n + 1 containing the values of the function through the series.
**Remarks**

Bessel.I[i] contains the value of the Bessel function of order i+xnu. The Bessel function \( I_v(x) \), is defined to be

\[
I_v(x) = \frac{1}{\pi} \int_0^\pi e^{x \cos \theta} \cos(v \theta) d \theta - \frac{\sin(v \pi)}{\pi} \int_0^\infty e^{-x \cosh t - vt} dt
\]

Here, argument xnu is represented by \( v \) in the above equation.

The input x must be nonnegative and less than or equal to \( \log(b) \) (b is the largest representable number).

The argument \( v = xnu \) must satisfy \( 0 \leq v \leq 1 \).

This function is based on a code due to Cody (1983), which uses backward recursion.

**J**

static public double[] J(double x, int n)

**Description**

Evaluates a sequence of Bessel functions of the first kind with integer order and real argument.

**Parameters**

- \( x \) – A double representing the argument for which the sequence of Bessel functions is to be evaluated.
- \( n \) – A int which specifies the order of the last element in the sequence.

**Returns**

A double array of length \( n + 1 \) containing the values of the function through the series.

**Remarks**

Bessel.J[i] contains the value of the Bessel function of order i at x for i = 0 to n. The Bessel function \( J_n(x) \), is defined to be

\[
J_n(x) = \frac{1}{\pi} \int_0^\pi \cos (x \sin \theta - n \theta) d \theta
\]

The algorithm is based on a code due to Sookne (1973b) that uses backward recursion with strict error control.

**J**

static public double[] J(double xnu, double x, int n)

**Description**

Evaluate a sequence of Bessel functions of the first kind with real order and real positive argument.

**Parameters**

- \( xnu \) – A double representing the lowest order desired. \( xnu \) must be at least zero and less than 1.
- \( x \) – A double representing the argument for which the sequence of Bessel functions is to be evaluated.
- \( n \) – A int representing the order of the last element in the sequence. If order is the highest order desired, set n to int(order).
Returns
A double array of length n+1 containing the values of the function through the series. Bessel.J[I] contains the value of the Bessel function of order I + v at x for I=0 to n.

Remarks
The Bessel function \( J_v(x) \), is defined to be

\[
J_v(x) = \frac{(x/2)^v}{\sqrt{\pi v (v + 1/2)}} \int_0^\pi \cos(x \cos \theta) \sin^{2v} \theta \, d\theta
\]

This code is based on the work of Gautschi (1964) and Skovgaard (1975). It uses backward recursion.

K
static public double[] K(double x, int n)

Description
Evaluates a sequence of modified Bessel functions of the third kind with integer order and real argument.

Parameters
- x – A double representing the argument for which the sequence of Bessel functions is to be evaluated.
- n – A int which specifies the order of the last element in the sequence.

Returns
A double array of length n+1 containing the values of the function through the series.

Remarks
This function uses \( e^rK_{\nu+k-1} \) for \( k = 1, \ldots, n \) and \( v = 0 \). For the definition of \( K_v(x) \), see below.

K
static public double[] K(double xnu, double x, int n)

Description
Evaluates a sequence of modified Bessel functions of the third kind with fractional order and real argument.

Parameters
- xnu – A double representing the fractional order of the function. xnu must be less than one in absolute value.
- x – A double representing the argument for which the sequence of Bessel functions is to be evaluated.
- n – A int representing the order of the last element in the sequence. If order is the highest order desired, set n to int(order).

Returns
A double array of length n+1 containing the values of the function through the series.
Remarks

Bessel.K[I] contains the value of the Bessel function of order I + v at x for I = 0 to n. The Bessel function $K_v(x)$ is defined to be

$$K_v(x) = \frac{\pi}{2} e^{x \pi i/2} [iJ_v(ix) - Y_v(ix)] \quad \text{for} \quad -\pi < \arg x \leq \frac{\pi}{2}$$

Currently, $xnu$ (represented by $v$ in the above equation) is restricted to be less than one in absolute value. A total of $n$ values is stored in the result, $K$.

$K[0] = K_v(x), K[1] = K_{v+1}(x), \ldots, K[n-1] = K_{v+n-1}(x)$.

This method is based on the work of Cody (1983).

**ScaledK**

static public double[] ScaledK(double v, double x, int n)

Description

Evaluate a sequence of exponentially scaled modified Bessel functions of the third kind with fractional order and real argument.

Parameters

- $v$ – A double representing the fractional order of the function. $v$ must be less than one in absolute value.
- $x$ – A double representing the argument for which the sequence of Bessel functions is to be evaluated.
- $n$ – A int representing the order of the last element in the sequence. If order is the highest order desired, set $n$ to int(order).

Returns

A double array of length $n+1$ containing the values of the function through the series.

Remarks

If $n$ is positive, Bessel.K[I] contains $e^x$ times the value of the Bessel function of order I + v at x for I = 0 to n.

If $n$ is negative, Bessel.K[I] contains $e^x$ times the value of the Bessel function of order v - I at x for I = 0 to n. This function evaluates $e^xK_{v+i-1}(x)$, for i=1,...,n where K is the modified Bessel function of the third kind. Currently, $v$ is restricted to be less than 1 in absolute value. A total of $|n| + 1$ elements are returned in the array. This code is particularly useful for calculating sequences for large $x$ provided $n = x$. (Overflow becomes a problem if $n < < x$.) $n$ must not be zero, and $x$ must be greater than zero. $|v|$ must be less than 1. Also, when $|n|$ is large compared with $x$, $|v + n|$ must not be so large that

$$e^xK_{v+n}(x) = e^x \frac{\Gamma(|v + n|)}{2(x/2)^{v+n}}$$

overflows. The code is based on work of Cody (1983).

**Y**

static public double[] Y(double xnu, double x, int n)

---

472 • Bessel

IMSL C# Numerical Library
Description
Evaluate a sequence of Bessel functions of the second kind with real nonnegative order and real positive argument.

Parameters

- `xnu` – A `double` representing the lowest order desired. `xnu` must be at least zero and less than 1.
- `x` – A `double` representing the argument for which the sequence of Bessel functions is to be evaluated.
- `n` – A `int` which specifies that `n + 1` elements will be evaluated in the sequence.

Returns
A `double` array of length `n + 1` containing the values of the function through the series.

Remarks
Bessel.K[I] contains the value of the Bessel function of order I + v at x for I=0 to n. The Bessel function $Y_v(x)$ is defined to be

$$
Y_v(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin \theta - v \theta) d \theta
$$

$$
- \frac{1}{\pi} \int_0^\infty \left[ e^{vt} + e^{-vt} \cos(v \pi) \right] e^{-x \sinh t} dt
$$

The variable `xnu` (represented by `v` in the above equation) must satisfy $0 \leq v < 1$. If this condition is not met, then `Y` is set to NaN. In addition, `x` must be in $[x_m, x_M]$ where $x_m = 6 \times 10^{-32}$ and $x_M = 16^9$. If $x < x_m$, then the largest representable number is returned; and if $x < x_M$, then zero is returned.

The algorithm is based on work of Cody and others, (see Cody et al. 1976; Cody 1969; NATS FUNPACK 1976). It uses a special series expansion for small arguments. For moderate arguments, an analytic continuation in the argument based on Taylor series with special rational minimax approximations providing starting values is employed. An asymptotic expansion is used for large arguments.

Example: The Bessel Functions
The Bessel functions I, J, and K are exercised for orders 0, 1, 2, and 3 at argument 10.e0.

```csharp
using System;
using Imsl.Math;

class BesselEx1
{
    public static void Main(string[] args)
    {
        double x = 10e0;
        int hiorder = 4;
        // Exercise some of the Bessel functions with argument 10.0
    }
}
```
double[] bi = Bessel.I(x, hiorder);
double[] bj = Bessel.J(x, hiorder);
double[] bk = Bessel.K(x, hiorder);

for (int i = 0; i < 4; i++)
{
    Console.Out.WriteLine(i + " " + bi[i] + " " + bj[i] + " " + bk[i]);
}
Console.Out.WriteLine();

Output
Order Bessel.I Bessel.J Bessel.K
0 2815.71662846626 -0.245935764451348 1.77800623161677E-05
1 2670.98830370126 0.0434727461688615 1.86487734538256E-05
2 2281.518967726 0.254630313685121 2.15098170069328E-05
3 1758.38071661085 0.0583793793051867 2.72527002565987E-05
Chapter 10: Miscellaneous

Types

class Complex ................................................................. 475
structure Physical ............................................................... 500
class EpsilonAlgorithm ......................................................... 513

Complex Structure

public structure Imsl.Math.Complex : System.IComparable, System.IFormattable

Set of mathematical functions for complex numbers. It provides the basic operations (addition, subtraction, multiplication, division) as well as a set of complex functions.

The binary operations have the form, where op is Add, Subtract, Multiply or Divide.

public static Complex op(Complex x, Complex y) // x op y
public static Complex op(Complex x, double y) // x op y
public static Complex op(double x, Complex y) // x op y

Complex objects are immutable. Once created there is no way to change their value. The functions in this class follow the rules for complex arithmetic as defined C9x Annex G: IEC 559-compatible complex arithmetic. The API is not the same, but handling of infinities, NaNs, and positive and negative zeros is intended to follow the same rules.

Field

I

public Imsl.Math.Complex I
Description
The imaginary unit.

Remarks
This constant is set to new Complex(0,1).

Constructors

Complex
public Complex(Imsl.Math.Complex z)

Description
Constructs a Complex equal to the argument.

Parameter
z – A Complex object.

Exception
System.NullReferenceException is thrown if z is null

Complex
public Complex(double re, double im)

Description
Constructs a Complex with real and imaginary parts given by the input arguments.

Parameters
re – A double value equal to the real part of the Complex object.
im – A double value equal to the imaginary part of the Complex object.

Complex
public Complex(double re)

Description
Constructs a Complex with a zero imaginary part.

Parameter
re – A double value equal to the real part of the Complex object.

Operators

Operator +
static public operator x + y
Description
Returns the sum of two Complex objects, $x+y$.

Parameters
- $x$ – A Complex object.
- $y$ – A Complex object.

Returns
A newly constructed Complex initialized to $x+y$.

Operator +
static public operator $x + y$

Description
Returns the sum of a Complex and a double, $x+y$.

Parameters
- $x$ – A Complex object.
- $y$ – A double value.

Returns
A newly constructed Complex initialized to $x+y$.

Operator +
static public operator $x + y$

Description
Returns the sum of a double and a Complex, $x+y$.

Parameters
- $x$ – A double value.
- $y$ – A Complex object.

Returns
A newly constructed Complex initialized to $x+y$.

Operator /
static public operator $x / y$

Description
Returns the result of a Complex object divided by a Complex object, $x/y$.

Parameters
- $x$ – A Complex object representing the numerator.
- $y$ – A Complex object representing the denominator.
Returns
A newly constructed Complex initialized to x/y.

Operator /  
static public operator x / y

Description
Returns the result of a Complex object divided by a double, x/y.

Parameters
   x – A Complex object representing the numerator.
   y – A double representing the denominator.

Returns
A newly constructed Complex initialized to x/y.

Operator /
static public operator x / y

Description
Returns the result of a double divided by a Complex object, x/y.

Parameters
   x – A double value.
   y – A Complex object representing the denominator.

Returns
A newly constructed Complex initialized to x/y.

Operator ==
static public operator x == y

Description
Returns true if x and y are equal.

Parameters
   x – A Complex value.
   y – A Complex value.

Returns
true if x and y are equal.

Operator ==
static public operator x == y

Description
Returns true if x and y are equal.
Parameters
  x – A Complex value.
  y – A double value.

Returns
true if x and y are equal.

Operator ==
static public operator x == y

Description
Returns true if x and y are equal.

Parameters
  x – A double value.
  y – A Complex value.

Returns
true if x and y are equal.

Operator !=
static public operator x != y

Description
Returns true if x and y are not equal.

Parameters
  x – A Complex value.
  y – A double value.

Returns
true if x and y are equal.

Operator !=
static public operator x != y

Description
Returns true if x and y are not equal.

Parameters
  x – A double value.
  y – A Complex value.

Returns
true if x and y are not equal.

Miscellaneous
Description
Returns true if x and y are not equal.

Parameters
x – A Complex value.
y – A Complex value.

Returns
ture if x and y are equal.

Operator *
static public operator x * y

Description
Returns the product of two Complex objects, x * y.

Parameters
x – A Complex object.
y – A Complex object.

Returns
A newly constructed Complex initialized to x × y.

Operator *
static public operator x * y

Description
Returns the product of a Complex object and a double, x * y.

Parameters
x – A Complex object.
y – A double value.

Returns
A newly constructed Complex initialized to x × y.

Operator *
static public operator x * y

Description
Returns the product of a double and a Complex object, x * y.

Parameters
x – A double value.
y – A Complex object.
Returns
A newly constructed Complex initialized to $x \times y$.

**Operator -**
static public operator $x - y$

**Description**
Returns the difference of two Complex objects, $x-y$.

**Parameters**
- $x$ – A Complex object.
- $y$ – A Complex object.

Returns
A newly constructed Complex initialized to $x-y$.

**Operator -**
static public operator $x - y$

**Description**
Returns the difference of a Complex object and a double, $x-y$.

**Parameters**
- $x$ – A Complex object.
- $y$ – A double value.

Returns
A newly constructed Complex initialized to $x-y$.

**Operator -**
static public operator $x - y$

**Description**
Returns the difference of a double and a Complex object, $x-y$.

**Parameters**
- $x$ – A double value.
- $y$ – A Complex object.

Returns
A newly constructed Complex initialized to $x-y$.

**Operator -**
static public operator $-x$

**Description**
Returns the negative of a Complex object, -$x$. 

**Miscellaneous**
Complex • 481
Parameter

- \( x \) – A Complex object.

Returns

A newly constructed Complex initialized to the negative of the Complex argument, \( x \).

Methods

Abs

\[
\text{static public double Abs(Imsl.Math.Complex z)}
\]

Description

Returns the absolute value (modulus) of a Complex, \( |z| \).

Parameter

- \( z \) – A Complex object.

Returns

A double value equal to the absolute value of the argument.

Acos

\[
\]

Description

Returns the inverse cosine (arc cosine) of a Complex, with branch cuts outside the interval \([-1,1]\) along the real axis.

Parameter

- \( z \) – A Complex object.

Returns

A newly constructed Complex initialized to the inverse (arc) cosine of the argument. The real part of the result is in the interval \([0, \pi]\).

Remarks

Specifically, if \( z = x + iy \),
\[
\overline{\text{acos}(z)} = \text{acos}(\overline{z}).
\]
\[
\text{acos}(\pm 0 + i0) \text{ returns } \pi/2 - i0.
\]
\[
\text{acos}(-\infty + i\infty) \text{ returns } 3\pi/4 - i\infty.
\]
\[
\text{acos}(+\infty + i\infty) \text{ returns } \pi/4 - i\infty.
\]
\[
\text{acos}(x + i\infty) \text{ returns } \pi/2 - i\infty, \text{ for finite } x.
\]
\[
\text{acos}(-\infty + iy) \text{ returns } \pi - i\infty, \text{ for positive-signed finite } y.
\]
acos(+∞ + iy) returns +0 − i∞, for positive-signed finite y.
acos(±∞ + iNaN) returns NaN ± i∞ (where the sign of the imaginary part of the result is unspecified).
acos(±0 + iNaN) returns π/2 + iNaN.
acos(NaN + i∞) returns NaN − i∞.
acos(x + iNaN) returns NaN + iNaN, for nonzero finite x.
acos(NaN + iy) returns NaN + iNaN, for finite y.
acos(NaN + iNaN) returns NaN + iNaN.

Acosh

Description
Returns the inverse hyperbolic cosine (arc cosh) of a Complex, with a branch cut at values less than one along the real axis.

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the inverse (arc) hyperbolic cosine of the argument. The real part of the result is non-negative and its imaginary part is in the interval [−iπ, iπ].

Remarks
Specifically, if z = x+iy,
acosh(¯z) = acosh(z).
acosh(±0 + i0) returns +0 + iπ/2.
acosh(−∞ + i∞) returns +∞ + i3π/4.
acosh(+∞ + i∞) returns +∞ + iπ/4.
acosh(x + i∞) returns +∞ + iπ/2, for finite x.
acosh(−∞ + iy) returns +∞ + iπ, for positive-signed finite y.
acosh(+∞ + iy) returns +∞ + i0, for positive-signed finite y.
acosh(NaN + i∞) returns +∞ + iNaN.
acosh(±∞ + iNaN) returns +∞ + iNaN.
acosh(x + iNaN) returns NaN + iNaN, for finite x.
acosh(NaN + iy) returns NaN + iNaN, for finite y.
acosh(NaN + iNaN) returns NaN + iNaN.

Add

Description
Returns the sum of two Complex objects, x+y.
Parameters
   x – A Complex object.
   y – A Complex object.

Returns
A newly constructed Complex initialized to x+y.

Add

Description
Returns the sum of a Complex and a double, x+y.

Parameters
   x – A Complex object.
   y – A double value.

Returns
A newly constructed Complex initialized to x+y.

Add
static public Imsl.Math.Complex Add(double x, Imsl.Math.Complex y)

Description
Returns the sum of a double and a Complex, x+y.

Parameters
   x – A double value.
   y – A Complex object.

Returns
A newly constructed Complex initialized to x+y.

Argument
static public double Argument(Imsl.Math.Complex z)

Description
Returns the argument (phase) of a Complex, in radians, with a branch cut along the negative real axis.

Parameter
   z – A Complex object.

Returns
A double value equal to the argument (or phase) of a Complex. It is in the interval $[-\pi, \pi]$.

Asin
Description
Returns the inverse sine (arc sine) of a Complex, with branch cuts outside the interval [-1,1] along the real axis.

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the inverse (arc) sine of the argument. The real part of the result is in the interval \([-\pi/2, +\pi/2]\).

Remarks
The value of Asin is defined in terms of the function Asinh, by \(\text{asin}(z) = -i \text{asinh}(iz)\).

Asinh

Description
Returns the inverse hyperbolic sine (arc sinh) of a Complex, with branch cuts outside the interval \([-i,i]\).

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the inverse (arc) hyperbolic sine of the argument. Its imaginary part is in the interval \([-i\pi/2, i\pi/2]\).

Remarks
Specifically, if \(z = x+iy\),
\(\text{asinh}(\bar{z}) = \overline{\text{asinh}(z)}\) and \(\text{asinh}\) is odd.
\(\text{asinh}(+0+0i)\) returns \(0+i0\).
\(\text{asinh}(\infty+i0)\) returns \(\infty+i\pi/4\).
\(\text{asinh}(x+i\infty)\) returns \(\infty+i\pi/2\) for positive-signed finite \(x\).
\(\text{asinh}(\infty+i0)\) returns \(\infty+i0\) for positive-signed finite \(y\).
\(\text{asinh}(\text{NaN}+i\infty)\) returns \(\pm\infty+i\text{NaN}\) (where the sign of the real part of the result is unspecified).
\(\text{asinh}(\text{NaN}+i0)\) returns \(\text{NaN}+i0\).
\(\text{asinh}(\text{NaN}+iy)\) returns \(\text{NaN}+i\text{NaN}\), for finite nonzero \(y\).
\(\text{asinh}(x+i\text{NaN})\) returns \(\text{NaN}+i\text{NaN}\), for finite \(x\).
\(\text{asinh}(\text{NaN}+i\text{NaN})\) returns \(\text{NaN}+i\text{NaN}\).

Atan
Description
Returns the inverse tangent (arc tangent) of a Complex, with branch cuts outside the interval \([-i, i]\) along the imaginary axis.

Parameter
- \(z\) – A Complex object.

Returns
A newly constructed Complex initialized to the inverse (arc) tangent of the argument. Its real part is in the interval \([-\pi/2, \pi/2]\).

Remarks
The value of Atan is defined in terms of the function Atanh, by \(\text{atan}(z) = -i\text{atanh}(iz)\).

Atanh

Description
Returns the inverse hyperbolic tangent (arc tanh) of a Complex, with branch cuts outside the interval \([-1, 1]\) on the real axis.

Parameter
- \(z\) – A Complex object.

Returns
A newly constructed Complex initialized to the inverse (arc) hyperbolic tangent of the argument. The imaginary part of the result is in the interval \([-i\pi/2, i\pi/2]\).

Remarks
Specifically, if \(z = x + iy\),
\(\text{atanh}(\bar{z}) = \text{atanh}(\bar{z})\) and atanh is odd.
\(\text{atanh}(+0 + i0)\) returns \(+0 + i0\).
\(\text{atanh}(+\infty + i0)\) returns \(+0 + i\pi/2\).
\(\text{atanh}(+\infty + iy)\) returns \(+0 + i\pi/2\), for finite positive-signed \(y\).
\(\text{atanh}(x + i\infty)\) returns \(+0 + i\pi/2\), for finite positive-signed \(x\).
\(\text{atanh}(+0 + iNaN)\) returns \(+0 + iNaN\).
\(\text{atanh}(NaN + i\infty)\) returns \(\pm0 + i\pi/2\) (where the sign of the real part of the result is unspecified).
\(\text{atanh}(+\infty + iNaN)\) returns \(+0 + iNaN\).
\(\text{atanh}(NaN + iy)\) returns \(NaN + iNaN\), for finite \(y\).
\(\text{atanh}(x + iNaN)\) returns \(NaN + iNaN\), for nonzero finite \(x\).
\(\text{atanh}(NaN + iNaN)\) returns \(NaN + iNaN\).

CompareTo
Final public int CompareTo(object obj)
Description
Compares this Complex to another Object.

Parameter
obj – An Object to be compared.

Returns
An int, 0 if obj is equal to this Complex; a value less than 0 if this Complex is less than obj; and a value greater than 0 if this Complex is greater than obj.

Remarks
If the Object is a Complex, this function behaves like compareTo(Complex). Otherwise, it throws a InvalidCastException (as Complex objects are comparable only to other Complex objects).

Exception
System.InvalidCastException is thrown if obj is not a Complex object

CompareTo
public int CompareTo(Imsl.Math.Complex z)

Description
Compares two Complex objects.

Parameter
z – A Complex to be compared.

Returns
The value 0 if z is equal to this Complex; a value less than 0 if this Complex is less than z; and a value greater than 0 if this Complex is greater than z.

Remarks
A lexigraphical ordering is used. First the real parts are compared in the sense of Double.compareTo. If the real parts are unequal this is the return value. If the return parts are equal then the comparison of the imaginary parts is returned.

Conjugate

Description
Returns the complex conjugate of a Complex object.

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the complex conjugate of Complex argument, z.

Cos
Description
Returns the cosine of a Complex.

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the cosine of the argument.

Remarks
The value of Cos is defined in terms of the function Cosh, by \( \cos(z) = \cosh(iz) \).

Cosh

Description
Returns the hyperbolic cosh of a Complex.

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the hyperbolic cosine of the argument.

Remarks
If \( z = x+iy \),
\( \cosh(\bar{z}) = \cosh(z) \) and \( \cosh \) is even.
\( \cosh(+0 + i0) \) returns \( 1 + i0 \).
\( \cosh(+0 + i\infty) \) returns \( \text{NaN} \pm i0 \) (where the sign of the imaginary part of the result is unspecified).
\( \cosh(+\infty + i0) \) returns \( +\infty + i0 \).
\( \cosh(+\infty + i\infty) \) returns \( +\infty + i\text{NaN} \).
\( \cosh(x + i\infty) \) returns \( \text{NaN} + i\text{NaN} \), for finite nonzero \( x \).
\( \cosh(+\infty + iy) \) returns \( +\infty[\cos(y) + i\sin(y)] \), for finite nonzero \( y \).
\( \cosh(+0 + i\text{NaN}) \) returns \( \text{NaN} \pm i0 \) (where the sign of the imaginary part of the result is unspecified).
\( \cosh(+\infty + i\text{NaN}) \) returns \( +\infty + i\text{NaN} \).
\( \cosh(x + i\text{NaN}) \) returns \( \text{NaN} + i\text{NaN} \), for finite nonzero \( x \).
\( \cosh(\text{NaN} + i0) \) returns \( \text{NaN} \pm i0 \) (where the sign of the imaginary part of the result is unspecified).
\( \cosh(\text{NaN} + iy) \) returns \( \text{NaN} + i\text{NaN} \), for all nonzero numbers \( y \).
\( \cosh(\text{NaN} + i\text{NaN}) \) returns \( \text{NaN} + i\text{NaN} \).

Divide
Description
Returns the result of a Complex object divided by a Complex object, \( x/y \).

Parameters
\( x \) – A Complex object representing the numerator.
\( y \) – A Complex object representing the denominator.

Returns
A newly constructed Complex initialized to \( x/y \).

Divide

Description
Returns the result of a Complex object divided by a double, \( x/y \).

Parameters
\( x \) – A Complex object representing the numerator.
\( y \) – A double representing the denominator.

Returns
A newly constructed Complex initialized to \( x/y \).

Divide

Description
Returns the result of a double divided by a Complex object, \( x/y \).

Parameters
\( x \) – A double value.
\( y \) – A Complex object representing the denominator.

Returns
A newly constructed Complex initialized to \( x/y \).

Equals
override public bool Equals(object x)

Description
Compares this object against the specified object.

Parameter
\( x \) – The object to compare with.
Returns
true if the objects are the same; false otherwise.

Exp

Description
Returns the exponential of a Complex z, exp(z).

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the exponential of the argument.

Remarks
Specifically, if z = x+iy,
exp(¯z) = exp(z).
exp(±0 + i0) returns 1 + i0.
exp(±∞ + i0) returns ±∞ + i0.
exp(−∞ + i∞) returns ±0 ± i0 (where the signs of the real and imaginary parts of the result are unspecified).
exp(±∞ + i∞) returns ±∞ + iNaN (where the sign of the real part of the result is unspecified).
exp(x + i∞) returns NaN + iNaN, for finite x.
exp(−∞ + iy) returns ±0[cos(y) + isin(y)], for finite y.
exp(±∞ + iy) returns ±∞[cos(y) + isin(y)], for finite nonzero y.
exp(−∞ + iNaN) returns ±0 ± i0 (where the signs of the real and imaginary parts of the result are unspecified).
exp(±∞ + iNaN) returns ±∞ + iNaN (where the sign of the real part of the result is unspecified).
exp(NaN + i0) returns NaN + i0.
exp(NaN + iy) returns NaN + iNaN, for all non-zero numbers y.
exp(x + iNaN) returns NaN + iNaN, for finite x.

GetHashCode
override public int GetHashCode()

Description
Returns a hashcode for this Complex.

Returns
A hash code value for this object.

Imag
public double Imag()
**Description**
Returns the imaginary part of a Complex object.

**Returns**
A double representing the imaginary part of a Complex object, z.

**Imag**
static public double Imag(Imsl.Math.Complex z)

**Description**
Returns the imaginary part of a Complex object.

**Parameter**
- z – A Complex object.

**Returns**
A double representing the imaginary part of the Complex object, z.

**Log**

**Description**
Returns the logarithm of a Complex z, with a branch cut along the negative real axis.

**Parameter**
- z – A Complex object.

**Returns**
A newly constructed Complex initialized to the logarithm of the argument. Its imaginary part is in the interval $[-i\pi, i\pi]$.

**Remarks**
Specifically, if $z = x+iy$,
\[ \log(\bar{z}) = \log(z). \]
\[ \log(0+0i) \text{ returns } -\infty + i\pi. \]
\[ \log(+0+i0) \text{ returns } -\infty + i0. \]
\[ \log(-\infty + i\infty) \text{ returns } +\infty + i3\pi/4. \]
\[ \log(+\infty + i\infty) \text{ returns } +\infty + i\pi/4. \]
\[ \log(x+i\infty) \text{ returns } +\infty + i\pi/2, \text{ for finite } x. \]
\[ \log(-\infty + iy) \text{ returns } +\infty + i\pi, \text{ for finite positive-signed } y. \]
\[ \log(+\infty + iy) \text{ returns } +\infty + i0, \text{ for finite positive-signed } y. \]
\[ \log(\pm\infty + iNaN) \text{ returns } +\infty + iNaN. \]
\[ \log(\text{NaN} + i\infty) \text{ returns } +\infty + i\text{NaN}. \]
\[ \log(x + i\text{NaN}) \text{ returns } \text{NaN} + i\text{NaN}, \text{ for finite } x. \]
log(NaN + iy) returns NaN + iNaN, for finite y.
log(NaN + iNaN) returns NaN + iNaN.

**Multiply**


**Description**

Returns the product of two Complex objects, x * y.

**Parameters**

- x – A Complex object.
- y – A Complex object.

**Returns**

A newly constructed Complex initialized to \(x \times y\).

**Multiply**


**Description**

Returns the product of a Complex object and a double, x * y.

**Parameters**

- x – A Complex object.
- y – A double value.

**Returns**

A newly constructed Complex initialized to \(x \times y\).

**Multiply**

static public Imsl.Math.Complex Multiply(double x, Imsl.Math.Complex y)

**Description**

Returns the product of a double and a Complex object, x * y.

**Parameters**

- x – A double value.
- y – A Complex object.

**Returns**

A newly constructed Complex initialized to \(x \times y\).

**MultiplyImag**

Description
Returns the product of a Complex object and a pure imaginary double, x * iy.

Parameters
- x – A Complex object.
- y – A double value representing a pure imaginary.

Returns
A newly constructed Complex initialized to x * iy.

MultiplyImag

Description
Returns the product of a pure imaginary double and a Complex object, ix * y.

Parameters
- x – A double value representing a pure imaginary.
- y – A Complex object.

Returns
A newly constructed Complex initialized to ix * y.

Negate

Description
Returns the negative of a Complex object, -z.

Parameter
- z – A Complex object.

Returns
A newly constructed Complex initialized to the negative of the Complex argument, z.

Parse
static public Imsl.Math.Complex Parse(string s)

Description
Converts the string representation of a number in a specified style to its Complex number equivalent.

Parameter
- s – A string containing a number to convert.
Returns
A Complex number represented in the String.

Parse
static public Imsl.Math.Complex Parse(string s, System.IFormatProvider formatProvider)

Description
Converts the string representation of a number in a specified style to its Complex number equivalent.

Parameters
- s – A string containing a number to convert.
- formatProvider – An IFormatProvider that supplies culture-specific formatting information about s.

Returns
A Complex number represented in the String.

Pow
static public Imsl.Math.Complex Pow(Imsl.Math.Complex z, double x)

Description
Returns the Complex z raised to the x power, with a branch cut for the first parameter (z) along the negative real axis.

Parameters
- z – A Complex object.
- x – A double value.

Returns
A newly constructed Complex initialized to z to the power x.

Pow

Description
Returns the Complex x raised to the Complex y power.

Parameters
- x – A Complex object.
- y – A Complex object.

Returns
A newly constructed Complex initialized to xy.
Remarks
The value of Pow is defined in terms of the functions Exp and Log, by pow(x,y) = exp(ylog(x)).

Real
public double Real()

Description
Returns the real part of a Complex object.

Returns
A double representing the real part of a Complex object, z.

Real
static public double Real(Imsl.Math.Complex z)

Description
Returns the real part of a Complex object.

Parameter
z – A Complex object.

Returns
A double representing the real part of the Complex object, z.

Sin

Description
Returns the sine of a Complex.

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the sine of the argument.

Remarks
The value of Sin is defined in terms of the function Sinh, by sin(z) = −i sinh(iz).

Sinh

Description
Returns the hyperbolic sine of a Complex.
**Parameter**

\[ z \] – A Complex object.

**Returns**
A newly constructed Complex initialized to the hyperbolic sine of the argument.

**Remarks**
If \( z = x + iy \),
\[
\sinh(\bar{z}) = \sinh(z) \text{ and } \sinh \text{ is odd.}
\]
\[
\sinh(+0 + i0) \text{ returns } +0 + i0.
\]
\[
\sinh(+\infty + i\infty) \text{ returns } \pm 0 + i\text{NaN} \text{ (where the sign of the real part of the result is unspecified).}
\]
\[
\sinh(-\infty + i\infty) \text{ returns } \pm 0 + i\text{NaN} \text{ (where the sign of the real part of the result is unspecified).}
\]
\[
\sinh(+\infty + iy) \text{ returns } +\infty[\cos(y) + i\sin(y)], \text{ for positive finite } y.
\]
\[
\sinh(x + i\infty) \text{ returns } \text{NaN} + i\text{NaN}, \text{ for positive finite } x.
\]
\[
\sinh(+0 + i\text{NaN}) \text{ returns } \pm 0 + i\text{NaN} \text{ (where the sign of the real part of the result is unspecified).}
\]
\[
\sinh(+\infty + i\text{NaN}) \text{ returns } \pm 0 + i\text{NaN} \text{ (where the sign of the real part of the result is unspecified).}
\]
\[
\sinh(x + i\text{NaN}) \text{ returns } \text{NaN} + i\text{NaN}, \text{ for finite nonzero } x.
\]
\[
\sinh(\text{NaN} + i0) \text{ returns } \text{NaN} + i0.
\]
\[
\sinh(\text{NaN} + iy) \text{ returns } \text{NaN} + i\text{NaN}, \text{ for all nonzero numbers } y.
\]
\[
\sinh(\text{NaN} + i\text{NaN}) \text{ returns } \text{NaN} + i\text{NaN}.
\]

**Sqrt**


**Description**
Returns the square root of a Complex, with a branch cut along the negative real axis.

**Parameter**

\[ z \] – A Complex object.

**Returns**
A newly constructed Complex initialized to square root of \( z \).

**Remarks**
Specifically, if \( z = x + iy \),
\[
\sqrt{z} = \sqrt{\bar{z}}.
\]
\[
\sqrt{\pm 0 + i0} \text{ returns } +0 + i0.
\]
\[
\sqrt{-\infty + iy} \text{ returns } +0 + i\infty, \text{ for finite positive-signed } y.
\]
\[
\sqrt{+\infty + iy} \text{ returns } +\infty + i0, \text{ for finite positive-signed } y.
\]
\[
\sqrt{x + i\infty} \text{ returns } +\infty + i\text{NaN}, \text{ for all } x \text{ (including NaN).}
\]
\[
\text{sqrt}(\infty + i\text{NaN}) \text{ returns } \text{NaN} \pm i\infty \text{ (where the sign of the imaginary part of the result is unspecified).}
\]
\[
\text{sqrt}(+\infty + i\text{NaN}) \text{ returns } +\infty + i\text{NaN}. 
\]
\[
\text{sqrt}(x + i\text{NaN}) \text{ returns } \text{NaN} + i\text{NaN} \text{ and optionally raises the invalid exception, for finite } x. 
\]
\[
\text{sqrt}(\text{NaN} + iy) \text{ returns } \text{NaN} + i\text{NaN} \text{ and optionally raises the invalid exception, for finite } y. 
\]
\[
\text{sqrt}(\text{NaN} + i\text{NaN}) \text{ returns } \text{NaN} + i\text{NaN}. 
\]

**Subtract**

\[
\]

**Description**

Returns the difference of two Complex objects, \(x - y\).

**Parameters**

- \(x\) – A Complex object.
- \(y\) – A Complex object.

**Returns**

A newly constructed Complex initialized to \(x - y\).

**Subtract**

\[
\text{static public Imsl.Math.Complex Subtract(Imsl.Math.Complex x, double y)}
\]

**Description**

Returns the difference of a Complex object and a double, \(x - y\).

**Parameters**

- \(x\) – A Complex object.
- \(y\) – A double value.

**Returns**

A newly constructed Complex initialized to \(x - y\).

**Subtract**

\[
\text{static public Imsl.Math.Complex Subtract(double x, Imsl.Math.Complex y)}
\]

**Description**

Returns the difference of a double and a Complex object, \(x - y\).

**Parameters**

- \(x\) – A double value.
- \(y\) – A Complex object.
Returns
A newly constructed Complex initialized to x-y.

Tan

Description
Returns the tangent of a Complex.

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the tangent of the argument.

Remarks
The value of Tan is defined in terms of the function Tanh, by \( \tan(z) = -i \tanh(iz) \).

Tanh

Description
Returns the hyperbolic tanh of a Complex.

Parameter
z – A Complex object.

Returns
A newly constructed Complex initialized to the hyperbolic tangent of the argument.

Remarks
If \( z = x+iy \),
\( \tanh(\bar{z}) = \overline{\tanh(z)} \) and tanh is odd.
\( \tanh(+0+i0) \) returns +0 + i0.
\( \tanh(+\infty + iy) \) returns 1 + i0, for all positive-signed numbers y.
\( \tanh(x+i\infty) \) returns NaN + iNaN, for finite x.
\( \tanh(+\infty + iNaN) \) returns 1 ± i0 (where the sign of the imaginary part of the result is unspecified).
\( \tanh(NaN+i0) \) returns NaN + i0.
\( \tanh(NaN+iy) \) returns NaN + iNaN, for all nonzero numbers y.
\( \tanh(x+iNaN) \) returns NaN + iNaN, for finite x.
\( \tanh(NaN+iNaN) \) returns NaN + iNaN.

ToString
override public string ToString()
Description
Returns a String representation for the specified Complex.

Returns
A String containing the Round-trip representation of this object. Round-trip guarantees that a numeric value converted to a String will be parsed back into the same numeric value.

ToString
Final public string ToString(string format, System.IFormatProvider formatProvider)

Description
Formats the value of the current instance using the specified format.

Parameters
format – The String specifying the format to use.
formatProvider – The IFormatProvider to use to format the value.

Example: Roots of a Quadratic Equation
The two roots of the quadratic equation \( ax^2 + bx + c \) are computed using the formula
\[
\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

using System;
using Imsl.Math;

public class ComplexEx1
{
    public static void Main(String[] args)
    {
        Complex a = new Complex(2.0, 3.0);
double b = 4.0;
Complex c = new Complex(1.0, -2.0);
Complex disc = Complex.Sqrt(b*b - 4.0*a*c);
Complex root1 = (-b + disc) / (2.0*a);
Complex root2 = (-b - disc) / (2.0*a);
    
Console.Out.WriteLine("Root1 = " + root1);
    Console.Out.WriteLine("Root2 = " + root2);
    
    
}

Output
Root1 = 0.19555270402037395+0.71433567154613054i
Root2 = -0.81093731940498925+0.20874125153079251i

Miscellaneous Complex • 499
**Physical Structure**

```csharp
public structure Imsl.Math.Physical
```

Return the value of various mathematical and physical constants.

The case of the `String` specifying the name of the physical constant does not matter. The names 'PI', 'Pi', 'pI' and 'pi' are equivalent.

The units of the physical constants are in SI units, (meter-kilogram-second).

The names allowed are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>Atomic mass unit</td>
<td>1.6605402E-27 kg</td>
<td>[1]</td>
</tr>
<tr>
<td>ATM</td>
<td>Standard atm pressure</td>
<td>1.01325E+5 N/m²</td>
<td>E[2]</td>
</tr>
<tr>
<td>AU</td>
<td>Astronomical unit</td>
<td>1.496E+11 m</td>
<td>[ ]</td>
</tr>
<tr>
<td>Avogadro</td>
<td>Avogadro’s number</td>
<td>6.0221367E+23 1/mole</td>
<td>[1]</td>
</tr>
<tr>
<td>Boltzman</td>
<td>Boltzman’s constant</td>
<td>1.380658E-23 J/K</td>
<td>[1]</td>
</tr>
<tr>
<td>C</td>
<td>Speed of light</td>
<td>2.997924580E+8 m/s</td>
<td>E[1]</td>
</tr>
<tr>
<td>Catalan</td>
<td>Catalan’s constant</td>
<td>0.915965...</td>
<td>E[3]</td>
</tr>
<tr>
<td>E</td>
<td>Base of natural logs</td>
<td>2.718...</td>
<td>E[3]</td>
</tr>
<tr>
<td>ElectronCharge</td>
<td>Electron change</td>
<td>1.60217733E-19 C</td>
<td>[1]</td>
</tr>
<tr>
<td>Euler</td>
<td>Euler’s constant gamma</td>
<td>0.577...</td>
<td>E[3]</td>
</tr>
<tr>
<td>Faraday</td>
<td>Faraday constant</td>
<td>9.6485309E+4 C/mole</td>
<td>[1]</td>
</tr>
<tr>
<td>FineStructure</td>
<td>Fine structure</td>
<td>7.29735308E-3</td>
<td>[1]</td>
</tr>
<tr>
<td>Gamma</td>
<td>Euler’s constant</td>
<td>0.577...</td>
<td>E[3]</td>
</tr>
<tr>
<td>Gas</td>
<td>Gas constant</td>
<td>8.314510 J/mole/K</td>
<td>[1]</td>
</tr>
<tr>
<td>Gravity</td>
<td>Gravitational constant</td>
<td>6.67259E-11 Nm²/kg²</td>
<td>[1]</td>
</tr>
<tr>
<td>Hbar</td>
<td>Planck constant / 2 pi</td>
<td>1.05457266E-34 J*s</td>
<td>[1]</td>
</tr>
<tr>
<td>PerfectGasVolume</td>
<td>Std vol ideal gas</td>
<td>2.241383E-2 m³/mole</td>
<td>[*]</td>
</tr>
<tr>
<td>Pi</td>
<td>Pi</td>
<td>3.141...</td>
<td>E[3]</td>
</tr>
<tr>
<td>Planck</td>
<td>Planck’s constant h</td>
<td>6.6260755E-34 J*s</td>
<td>[1]</td>
</tr>
<tr>
<td>ProtonMass</td>
<td>Proton mass</td>
<td>1.6726231E-27 kg</td>
<td>[1]</td>
</tr>
<tr>
<td>Rydberg</td>
<td>Rydberg’s constant</td>
<td>1.0973731534E+7 /m</td>
<td>[1]</td>
</tr>
<tr>
<td>SpeedLight</td>
<td>Speed of light</td>
<td>2.997924580E+8 m/s</td>
<td>E[1]</td>
</tr>
<tr>
<td>StandardGravity</td>
<td>Standard g</td>
<td>9.80665 m/s²</td>
<td>E[2]</td>
</tr>
<tr>
<td>StandardPressure</td>
<td>Standard atm pressure</td>
<td>1.01325E+5 N/m²</td>
<td>E[2]</td>
</tr>
<tr>
<td>StefanBoltzmann</td>
<td>Stefan-Boltzman</td>
<td>5.67051E-8 W/K⁴/m⁴</td>
<td>[1]</td>
</tr>
<tr>
<td>WaterTriple</td>
<td>Triple point of water</td>
<td>2.7316E+2 K</td>
<td>E[2]</td>
</tr>
</tbody>
</table>

The reference for constants are indicated by the code in the [] comment above.
The constants marked with an E before the [] are exact (to machine precision).

Units strings have the form U₁*U₂*...*Uₘ/V₁/.../Vₙ, where Uᵢ and Vᵢ are the names of basic units or are the names of basic units raised to a power. Examples are, 'METER*KILOGRAM/SECOND', 'M*KG/S', 'METER', or 'M/KG²'. These strings are case insensitive.

The basic unit names allowed are as follows.

Units of time

day, hour = hr, min = minute, s = sec = second, year

Units of frequency

Hertz = Hz

Units of mass

AMU, g = gram, lb = pound, ounce = oz, slug

Units of distance

Angstrom, AU, ft = feet = foot, in = inch, m = meter = metre, micron, mile, mill, parsec, yard

Units of area

acre

Units of volume

l = liter = litre

Units of force

dyne, N = Newton, poundal

Units of energy

BTU(thermochemical), Erg, J = Joule

Units of work

W = watt

Units of pressure

ATM = atmosphere, bar, Pascal

Units of temperature

degC = Celsius, degF = Fahrenheit, degK = Kelvin

Units of viscosity

poise, stoke
Units of charge
Abcoulomb, C = Coulomb, statcoulomb
Units of current
A = ampere, abampere, statampere
Units of voltage
Abvolt, V = volt
Units of magnetic induction
T = Tesla, Wb = Weber
Other units
1, farad, mole, Gauss, Henry, Maxwell, Ohm

The following metric prefixes may be used with the above units. Note that the one or two letter prefixes may only be used with one letter unit abbreviations.

A = atto = 1.E-18
F = femto = 1.E-15
P = pico = 1.E-12
N = nano = 1.E-9
U = micro = 1.E-6
M = milli = 1.E-3
C = centi = 1.E-2
D = deci = 1.E-1
DK = deca = 1.E+1
K = kilo = 1.E+3
myria = 1.E+4 (no single letter prefix; M means milli)
mega = 1.E+6 (no single letter prefix; M means milli)
G = giga = 1.E+9
T = tera = 1.E+12

Constructors

Physical
public Physical(double magnitude, string units)
Description
Constructs a new Physical object and initializes this object to a double value.

Parameters
magnitude – A double value to which the copy of the object is initialized.
units – A String specifying the unit.

Physical
public Physical(double magnitude, int length, int mass, int time, int current, int temperature)

Description
Constructs a new Physical object and initializes this object to a double value along with int values for length, mass, time, current, and temperature.

Parameters
magnitude – A double value to which this object is initialized.
length – An int value assigned to this object’s length.
mass – An int value assigned to this object’s mass.
time – An int value assigned to this object’s time.
current – An int value assigned to this object’s current.
temperature – An int value assigned to this object’s temperature.

Operators

Operator +
static public operator x + y

Description
Add two compatible Physical objects.

Exception
System.ArgumentException is thrown if x and y are not compatible

Parameters
x – A Physical object which is to be added.
y – A Physical object which is to be added.

Returns
A Physical object which is the sum of x + y.

Operator /
static public operator x / y
Description
Divide two Physical objects.

Parameters
- \( x \) – A Physical object which is the numerator.
- \( y \) – A Physical object which is the divisor.

Returns
A Physical object which is the result of \( x/y \).

Operator /
static public operator x / y

Description
Divide a Physical object by a double.

Parameters
- \( x \) – A Physical object which is the numerator.
- \( y \) – A double object which is the divisor.

Returns
A Physical object which is the result of \( x/y \).

Operator /
static public operator x / y

Description
Divide a double by a Physical object.

Parameters
- \( x \) – A double which is the numerator.
- \( y \) – A Physical object which is the divisor.

Returns
A Physical object which is the result of \( x/y \).

Operator ==
static public operator x == y

Description
Returns true if \( x \) and \( y \) are equal.

Parameters
- \( x \) – A Physical object.
- \( y \) – A Physical object.
Returns
A bool value of true if x and y are equal.

Operator ()
static public operator () x

Description
Returns the value of this dimensionless object.

Exception
System.ArgumentException is thrown if the this object is not dimensionless

Parameter
x – A Physical object.

Returns
The double value of the dimensionless object.

Operator !=
static public operator x != y

Description
Returns true if x and y are not equal.

Parameters
x – A Physical object.
y – A Physical object.

Returns
The double value of the dimensionless object.

Operator *
static public operator x * y

Description
Multiply two Physical objects.

Parameters
x – A Physical object which is to be multiplied.
y – A Physical object which is to be multiplied.

Returns
A Physical object which is the product of x and y.
**Description**
Multiply a Physical object and a double.

**Parameters**
- x – A Physical object which is to be multiplied.
- y – A double which is to be multiplied.

**Returns**
A Physical object which is the product of x and y.

**Operator ***
static public operator x * y

**Description**
Multiply a double and a Physical object

**Parameters**
- x – A double which is to be multiplied.
- y – A Physical object which is to be multiplied.

**Returns**
A Physical object which is the product of x and y.

**Operator -**
static public operator x - y

**Description**
Subtract two compatible Physical objects.

**Exception**
System.ArgumentException is thrown if x and y are not compatible

**Parameters**
- x – A Physical object.
- y – A Physical object which is to be subtracted from x.

**Returns**
A Physical object which is the result of x - y.

**Operator -**
static public operator - x

**Description**
Negate a Physical object.

**Parameter**
- x – A Physical object which is to be negated.
Returns
A Physical object which has been negated.

Methods

Add

Description
Adds two compatible Physical objects.

Parameters
  x – A Physical object which is to be added.
  y – A Physical object which is to be added.

Returns
A Physical object which is the sum of x + y.

Exception
  System.ArgumentException is thrown if x and y are not compatible

CheckCompatibility

Description
Checks the compatibility of two Physical objects.

Parameters
  x – A Physical object.
  y – A Physical object to be checked against x.

Exception
  System.ArgumentException is thrown if the two Physical objects are incompatible

Constant
static public Imsl.Math.Physical Constant(string name)

Description
Returns the value of a constant, given its name.

Parameter
  name – A String representing the name of the constant to be returned.
Returns
The Physical object containing the value of the constant, in its default units.

Exception
System.ArgumentException is thrown when the name given is undefined

Constant
static public double Constant(string name, string units)

Description
Returns the value of a constant, given its name, in the specified units.

Parameters
name – A String representing the name of the constant to be returned.
units – A String representing the units in which the constant is to be returned.

Returns
A double containing the value of the constant in the specified units.

Exception
System.ArgumentException is thrown if the constant name is undefined

Convert

Description
Converts a value to a different set of units.

Parameters
physical – A Physical object specifying the value to be converted.
unitsNew – A String specifying the units to which physical is to be converted.

Returns
A Physical object containing the value of physical converted to the new units.

Exception
System.ArgumentException is thrown if the new and old units are incompatible

DefineConstant
static public void DefineConstant(string name, Imsl.Math.Physical magnitude)

Description
Defines a new constant.
Parameters

- name – A String specifying the name of the constant to be defined.
- magnitude – A Physical object defining the value of the new constant.

**DefinePrefix**

```csharp
static public void DefinePrefix(string name, double magnitude)
```

**Description**

Defines a new prefix.

**Parameters**

- name – A String specifying the name of the prefix to be defined.
- magnitude – The double value of the prefix.

**DefineUnit**

```csharp
static public void DefineUnit(string name, Imsl.Math.Physical magnitude)
```

**Description**

Defines a new unit.

**Parameters**

- name – A String specifying the name of the unit to be defined.
- magnitude – A Physical object defining the value of one unit in terms of SI units.

**Divide**

```csharp
```

**Description**

Divides two Physical objects.

**Parameters**

- x – A Physical object which is the numerator.
- y – A Physical object which is the divisor.

**Returns**

A Physical object which is the result of x/y.

```csharp
```

**Description**

Divides a Physical object by a double.

**Parameters**

- x – A Physical object which is the numerator.
- y – A double object which is the divisor.
Returns
A Physical object which is the result of x/y.

Divide
static public Imsl.Math.Physical Divide(double x, Imsl.Math.Physical y)

Description
Divides a double by a Physical object.

Parameters
  x – A double which is the numerator.
  y – A Physical object which is the divisor.

Returns
A Physical object which is the result of x/y.

DoubleValue
public double DoubleValue()

Description
Returns the value of this dimensionless object.

Returns
The double value of the dimensionless object.

Exception
  System.ArgumentException is thrown if the this object is not dimensionless

Equals
override public bool Equals(object x)

Description
Returns true if x equals this value.

Parameter
  x – An Object to be tested for equality.

Returns
A bool value of true if x and this Physical object are equal.

Equals

Description
Returns true if x and y are equal.
Parameters

- \( x \) – A Physical object.
- \( y \) – A Physical object.

Returns

A bool value of true if \( x \) and \( y \) are equal.

**GetHashCode**

**Description**

Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.

**Returns**

A hash code for the current Object.

**Multiply**


**Description**

Multiply two Physical objects.

**Parameters**

- \( x \) – A Physical object which is to be multiplied.
- \( y \) – A Physical object which is to be multiplied.

**Returns**

A Physical object which is the product of \( x \) and \( y \).

**Multiply**

static public Imsl.Math.Physical Multiply(Imsl.Math.Physical \( x \), double \( y \))

**Description**

Multiply a Physical object and a double.

**Parameters**

- \( x \) – A Physical object which is to be multiplied.
- \( y \) – A double which is to be multiplied.

**Returns**

A Physical object which is the product of \( x \) and \( y \).

**Multiply**

static public Imsl.Math.Physical Multiply(double \( x \), Imsl.Math.Physical \( y \))

Miscellaneous
**Description**
Multiply a double and a Physical object.

**Parameters**
- x – A double which is to be multiplied.
- y – A Physical object which is to be multiplied.

**Returns**
A Physical object which is the product of x and y.

---

**Negate**

**Description**
Negate a Physical object.

**Parameter**
- x – A Physical object which is to be negated.

**Returns**
A Physical object which has been negated.

---

**Subtract**

**Description**
Subtract two compatible Physical objects.

**Parameters**
- x – A Physical object.
- y – A Physical object which is to be subtracted from x.

**Returns**
A Physical object which is the result of x - y.

**Exception**
System.ArgumentException is thrown if x and y are not compatible

---

**ToString**
override public string ToString()

**Description**
Returns a String containing the value and units, if any.
Returns
A String specifying the value and units, if any, of this Physical object.

UnitsString
public string UnitsString()

Description
Returns a String containing the units only.

Returns
A String specifying the units of this Physical object.

Example: Compute Kinetic Energy
The kinetic energy of a mass in motion is given by

\[ T = \frac{1}{2}mv^2 \]

where \( m \) is the mass and \( v \) is the velocity. In this example the mass is 2.4 pounds and the velocity is 6.7 meters per second. The infix operators defined by Physical automatically handle the unit conversions and computes the current units for the result.

```csharp
using System;
using Imsl.Math;

public class PhysicalEx1
{
    public static void Main(String[] args)
    {
        Physical mass = new Physical(2.4, "pound");
        Physical velocity = new Physical(6.7, "m/s");
        Physical energy = 0.5*mass*velocity*velocity;
        Console.Out.WriteLine("Kinetic energy is " + energy);
    }
}
```

Output
Kinetic energy is 24.43411378716 m^2*kg/s^2

EpsilonAlgorithm Class

public class Imsl.Math.EpsilonAlgorithm
The class is used to determine the limit of a sequence of approximations, by means of the Epsilon algorithm of P. Wynn.
An estimate of the absolute error is also given. The condensed Epsilon table is computed. Only those elements needed for the computation of the next diagonal are preserved.

**Property**

**ErrorEstimate**  
public double ErrorEstimate {get; }

**Description**  
Returns the current error estimate.

**Property Value**  
A double containing the current error estimate.

**Constructors**

**EpsilonAlgorithm**  
public EpsilonAlgorithm()

**Description**  
The class is used to determine the limit of a sequence of approximations, by means of the Epsilon algorithm of P. Wynn.

**Remarks**  
An estimate of the absolute error is also given. The condensed Epsilon table is computed. Only those elements needed for the computation of the next diagonal are preserved.

**EpsilonAlgorithm**  
public EpsilonAlgorithm(int maxTableSize)

**Description**  
The class is used to determine the limit of a sequence of approximations, by means of the Epsilon algorithm of P. Wynn.

**Parameter**  
maxTableSize – A int which specifies the maximum size of Epsilon Table to be computed.

**Remarks**  
An estimate of the absolute error is also given. The condensed Epsilon table is computed. Only those elements needed for the computation of the next diagonal are preserved.
Method

Extrapolate
public double Extrapolate(double x)

Description
Extrapolates the convergence limit of a sequence.

Parameter
x – A double which specifies the next point in the original series.

Returns
A double containing the estimate of the limit of the series.

Example: The Epsilon Algorithm

The Epsilon algorithm is used to accelerate a series of partial sums of an infinite sum. The error shows that the Epsilon algorithm is effective.

```csharp
using System;
using Imsl.Math;

class EpsilonAlgorithmEx1
{
    static void Main(string[] arg)
    {
        EpsilonAlgorithm eps = new EpsilonAlgorithm();

        int n = 100;
        double sum = 0.0;
        for (int i = 1; i < n; i++)
        {
            sum += 1.0 / (i * i);
            eps.Extrapolate(sum);
        }
        sum += 1.0 / (n * n);
        double extrapolated = eps.Extrapolate(1.0 / (n * n));
        double expected = System.Math.PI * System.Math.PI / 6.0;
        double exError = expected - extrapolated;
        double sumError = expected - sum;
        Console.WriteLine("Expected {0:0.00000}", expected);
        Console.WriteLine("Extrapolated {0:0.00000} error = {1:0.00000}",
                         extrapolated, exError);
        Console.WriteLine("Sum {0:0.00000} error = {1:0.00000}",
                         sum, sumError);
    }
}
```

Output

Expected 1.64493

Miscellaneous EpsilonAlgorithm • 515
Extrapolated 1.64278  error = 0.00216
Sum       1.63498  error = 0.00995
Chapter 11: Printing Functions

Types

class PrintMatrix .......................................................... 517
class PrintMatrixFormat ................................................... 522
class PrintMatrixFormat.ParsePosition ................................ 526
enumeration PrintMatrix.MatrixType .................................. 527
enumeration PrintMatrixFormat.FormatType .......................... 528
enumeration PrintMatrixFormat.ColumnLabelType ................... 530
enumeration PrintMatrixFormat.RowLabelType ....................... 531

PrintMatrix Class

public class Imsl.Math.PrintMatrix

Matrix printing utilities.

Constructors

PrintMatrix

public PrintMatrix()

Description

Creates an instance of the PrintMatrix class without a title and directs it to the default output stream.

Remarks

The matrix is printed without a title to System.Console.Out.
**PrintMatrix**

public PrintMatrix(System.IO.TextWriter writer)

**Description**

Creates an instance of the PrintMatrix class without a title and directs it to a specified output stream.

**Parameter**

writer – The TextWriter to which the matrix is to be written.

**PrintMatrix**

public PrintMatrix(string title)

**Description**

Creates a PrintMatrix object with a title directed to the default output stream.

**Parameter**

title – A String which specifies the title to be printed above the matrix.

**Remarks**

The matrix is printed without a title to System.Console.Out.

**PrintMatrix**

public PrintMatrix(System.IO.TextWriter writer, string title)

**Description**

Creates a PrintMatrix object with a title directed to a specified output stream.

**Parameters**

writer – A String which specifies the TextWriter to which the matrix is to be written.

title – The title to be printed above the matrix.

**Methods**

**Print**

virtual public void Print(string text)

**Description**

Prints a string.

**Parameter**

text – The String to be printed.
Remarks
This function can be overridden to print to something other than a PrintStream.

Print
virtual public void Print(object array)

Description
Prints an nRow by nColumn matrix with the default format.

Parameter
array – A two-dimensional, non-empty, rectangular Object array.

Print
virtual public void Print(Imsl.Math.PrintMatrixFormat pmf, object array)

Description
Prints an nRow by nColumn matrix with specified format.

Parameters
pmf – A PrintMatrixFormat matrix format.
array – A two-dimensional, non-empty, rectangular Object array.

PrintHTML
public void PrintHTML(Imsl.Math.PrintMatrixFormat pmf, object array, int nRows, int nColumns)

Description
Prints an nRow by nColumn matrix with specified format for HTML output.

Parameters
pmf – A PrintMatrixFormat matrix format.
array – The Matrix to be printed.
nRows – An int specifying the number of rows in the matrix.
nColumns – An int specifying the number of columns in the matrix.

Println
virtual public void Println()

Description
Prints a newline.

Remarks
This function can be overridden to print to something other than a PrintStream.

SetColumnSpacing
public Imsl.Math.PrintMatrix SetColumnSpacing(int columnSpacing)
Sets the number of spaces between columns.

**Parameter**

columnSpacing – An int specifying the number of spaces between columns.

**Returns**
The PrintMatrix object.

**Remarks**
The default value is 2.

**SetEqualColumnWidths**

```
public Imsl.Math.PrintMatrix SetEqualColumnWidths(bool equalColumnWidths)
```

**Description**
Force all of the columns to have the same width.

**Parameter**

equalColumnWidths – A bool which specifies that all column widths will be equal.

**Returns**
The PrintMatrix object.

**SetMatrixType**

```
public Imsl.Math.PrintMatrix SetMatrixType(Imsl.Math.PrintMatrix.MatrixType matrixType)
```

**Description**
Set matrix type.

**Parameter**

matrixType – An int specifying the matrix type.

**Returns**
The PrintMatrix object.

**Remarks**
Values for matrixType are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enumeration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MatrixType.Full</td>
</tr>
<tr>
<td>1</td>
<td>MatrixType.UpperTriangular</td>
</tr>
<tr>
<td>2</td>
<td>MatrixType.LowerTriangular</td>
</tr>
<tr>
<td>3</td>
<td>MatrixType.StrictUpperTriangular</td>
</tr>
<tr>
<td>4</td>
<td>MatrixType.StrictLowerTriangular</td>
</tr>
</tbody>
</table>

**SetPageWidth**

```
public Imsl.Math.PrintMatrix SetPageWidth(int pageWidth)
```

520 • PrintMatrix

IMSL C# Numerical Library
Description
Sets the page width.

Parameter
pageWidth – An int specifying the page width.

Returns
The PrintMatrix object.

Remarks
The default value is the largest possible integer.

SetTitle
public Imsl.Math.PrintMatrix SetTitle(string title)

Description
Sets the matrix title.

Parameter
title – A String specifying the title of the matrix.

Returns
The PrintMatrix object.

Example: Matrix and PrintMatrix
The 1 norm of a matrix is found using a method from the Matrix class. The matrix is printed using the
PrintMatrix class.

using System;
using Imsl.Math;

public class PrintMatrixEx1
{
    public static void Main(String[] args)
    {
        double nrm1;
        double[,] a = {{0.0, 1.0, 2.0, 3.0},
                       {4.0, 5.0, 6.0, 7.0},
                       {8.0, 9.0, 8.0, 1.0},
                       {6.0, 3.0, 4.0, 3.0}};

        // Get the 1 norm of matrix a
        nrm1 = Matrix.OneNorm(a);

        // Construct a PrintMatrix object with a title
        PrintMatrix p = new PrintMatrix("A Simple Matrix");

        // Print the matrix and its 1 norm
        p.Print(a);
        Console.Out.WriteLine("The 1 norm of the matrix is " + nrm1);
Output

A Simple Matrix

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

The 1 norm of the matrix is 20

PrintMatrixFormat Class

```csharp
public class Imsl.Math.PrintMatrixFormat

This class can be used to customize the actions of PrintMatrix.
By default, entries are formatted using the data type’s ToString method.
```

See Also

Imsl.Math.PrintMatrix (p. 517)

Properties

**FirstColumnNumber**
```csharp
public int FirstColumnNumber {get; set; }
```

Description

Turns on column labeling with index numbers and sets the index for the label of the first column.

Property Value

The number for the first column label.

Remarks

This is usually 0 or 1. The default is 0.

**FirstRowNumber**
```csharp
public int FirstRowNumber {get; set; }
```
**Description**

Turns on row labeling with index numbers and sets the index for the label of the first row.

**Property Value**

The number for the first row label.

**Remarks**

This is usually 0 or 1. The default is 0.

---

**NumberFormat**

`public string NumberFormat {get; set; }`

**Description**

The NumberFormat to be used in formatting `double` and `Complex` entries.

**Property Value**

A `String` containing the format to be used in Complex entries.

---

**Constructor**

`public PrintMatrixFormat()`

**Description**

Constructs a `PrintMatrixFormat` object.

---

**Methods**


**Description**

Returns a formatted string. This method is used by the methods

Imsl.Math.PrintMatrix.Print(System.String) (p. 518) and


**Parameters**

- `type` – The type of string requested. See `PrintMatrixFormat.FormatType` Enumeration.
- `entry` – The entry to be formatted. This is only used if type equals `Imsl.Math.PrintMatrixFormat.FormatType.Entry` (p. 530). For other values of type, this can be set to null.
row – The (0-based) row number of the element to be formatted. This is -1 if there is no row number associated with this request.

col – The (0-based) column number of the element to be formatted. This is -1 if there is no column number associated with this request.

pos – A ParsePosition object used to indicate the alignment center of the return string. This is used only if type is Imsl.Math.PrintMatrixFormat.FormatType.Entry (p. 530).

Returns
A String to be put into the printed table.

Remarks
Note, if type is not FormatType.Entry, pos will be set based on the following criteria.

<table>
<thead>
<tr>
<th>entry</th>
<th>behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>The index is the position of the decimal point.</td>
</tr>
<tr>
<td>int</td>
<td>The index is the position of the end of the formatted integer.</td>
</tr>
</tbody>
</table>

See Also: Imsl.Math.PrintMatrixFormat.FormatType (p. 528)

SetColumnLabels
public void SetColumnLabels(string[] columnLabels)

Description
Turns on column labeling using the given labels.

Parameter

columnLabels – An array of Strings to be used as column labels. If there are more columns than labels, the labels are reused.

SetNoColumnLabels
virtual public void SetNoColumnLabels()

Description
Turns off column labels.

SetNoRowLabels
virtual public void SetNoRowLabels()

Description
Turns off row labels.

Example: Matrix Formatting
A simple matrix is printed using the default format with the PrintMatrix class. The PrintMatrixFormat class is then used to change the default format.
using System;
using Imsl.Math;

public class PrintMatrixFormatEx1
{
    public static void Main(String[] args)
    {
        double[,] a = {
            {0.0, 1.0, 2.0, 3.0},
            {4.0, 5.0, 6.0, 7.0},
            {8.0, 9.0, 8.0, 1.0},
            {6.0, 3.0, 4.0, 3.0}};

        // Construct a PrintMatrix object with a title
        PrintMatrix p = new PrintMatrix("A Simple Matrix");

        // Print the matrix
        p.Print(a);

        // Turn row and column labels off
        PrintMatrixFormat mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();

        // Print the matrix
        p.Print(mf, a);
    }
}

Output
A Simple Matrix
0 1 2 3
0 0 1 2 3
1 4 5 6 7
2 8 9 8 1
3 6 3 4 3

A Simple Matrix
0 1 2 3
4 5 6 7
8 9 8 1
6 3 4 3

Example: Matrix Formatting
A matrix is printed in CSV (comma separated value) format. This is done by overriding the format method of PrintMatrixFormat to add commas after all but the last number in each row.

using System;
using Imsl.Math;

public class PrintMatrixFormatEx2 : PrintMatrixFormat

Printing Functions

PrintMatrixFormat • 525
```csharp
{ 
    private int ncols;

    public PrintMatrixFormatEx2(int ncols)
    {
        this.ncols = ncols;
    }

    public override String Format(FormatType type, System.Object entry, int row,
        int col, ParsePosition pos)
    {
        String text = base.Format(type, entry, row, col, pos);
        if (type == FormatType.Entry)
        {
            if (col < ncols - 1)
                text += ",";
        }
        return text;
    }

    public static void Main(System.String[] args)
    {
        double[][] a = {
            new double[]{0.0, 1.0, 2.0},
            new double[]{4.0, 5.0, 6.0},
            new double[]{8.0, 9.0, 8.0},
            new double[]{6.0, 3.0, 4.0}
        };

        PrintMatrixFormat mf = new PrintMatrixFormatEx2(3);
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();

        // Print the matrix
        new PrintMatrix().Print(mf, a);
    }
}

Output
0, 1, 2
4, 5, 6
8, 9, 8
6, 3, 4

PrintMatrixFormat.ParsePosition Class

public class Imsl.Math.PrintMatrixFormat.ParsePosition

526 • PrintMatrixFormat.ParsePosition  IMSL C# Numerical Library
Tracks the current position during parsing.

**Property**

**Index**

```csharp
public int Index {get; set; }
```

**Description**

Current parse position.

**Property Value**

An int containing the current index position.

**Constructor**

**ParsePosition**

```csharp
public ParsePosition(int index)
```

**Description**

Creates a ParsePosition.

**Parameter**

- `index` – The initial position.

---

**PrintMatrix.MatrixType Enumeration**

```csharp
public enumeration Imsl.Math.PrintMatrix.MatrixType
```

MatrixType indicates what part of the matrix is to be printed.

**Fields**

**Full**

```csharp
public Imsl.Math.PrintMatrix.MatrixType Full
```
Description
Indicates that the full matrix is to be printed.

**LowerTriangular**

```csharp
public Imsl.Math.PrintMatrix.MatrixType LowerTriangular
```

Description
Indicates that only the lower triangular elements of the matrix are to be printed. The matrix still must be a rectangular matrix.

**StrictLowerTriangular**

```csharp
public Imsl.Math.PrintMatrix.MatrixType StrictLowerTriangular
```

Description
Indicates that only the strict lower triangular elements of the matrix are to be printed. The matrix still must be a rectangular matrix.

**StrictUpperTriangular**

```csharp
public Imsl.Math.PrintMatrix.MatrixType StrictUpperTriangular
```

Description
Indicates that only the strict upper triangular elements of the matrix are to be printed. The matrix still must be a rectangular matrix.

**UpperTriangular**

```csharp
public Imsl.Math.PrintMatrix.MatrixType UpperTriangular
```

Description
Indicates that only the upper triangular elements of the matrix are to be printed. The matrix still must be a rectangular matrix.

---

**PrintMatrixFormat.FormatType Enumeration**

```csharp
public enumeration Imsl.Math.PrintMatrixFormat.FormatType
```

FormatType specifies the argument to format.

---

**Fields**

**BeginColumnLabel**

```csharp
public Imsl.Math.PrintMatrixFormat.FormatType BeginColumnLabel
```
Description
Indicates that the formatting string for ending a column label is to be returned.

**BeginColumnLabels**
public Imsl.Math.PrintMatrixFormat.FormatType BeginColumnLabels

Description
Indicates that the formatting string for beginning a column label row is to be returned.

**BeginEntry**
public Imsl.Math.PrintMatrixFormat.FormatType BeginEntry

Description
Indicates that the formatted string for beginning an entry is to be returned.

**BeginMatrix**
public Imsl.Math.PrintMatrixFormat.FormatType BeginMatrix

Description
Indicates that the formatting string for beginning a matrix is to be returned.

**BeginRow**
public Imsl.Math.PrintMatrixFormat.FormatType BeginRow

Description
Indicates that the formatting string for beginning a row is to be returned.

**BeginRowLabel**
public Imsl.Math.PrintMatrixFormat.FormatType BeginRowLabel

Description
Indicates that the formatting string for beginning a row label is to be returned.

**ColumnLabel**
public Imsl.Math.PrintMatrixFormat.FormatType ColumnLabel

Description
Indicates that the formatted string for a given column label is to be returned.

**EndColumnLabel**
public Imsl.Math.PrintMatrixFormat.FormatType EndColumnLabel

Description
Indicates that the formatting string for ending a column label is to be returned.

**EndColumnLabels**
public Imsl.Math.PrintMatrixFormat.FormatType EndColumnLabels

---

**Printing Functions**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BeginColumnLabels</code></td>
<td>Indicates the formatting string for beginning a column label row is to be returned.</td>
</tr>
<tr>
<td><code>BeginEntry</code></td>
<td>Indicates that the formatted string for beginning an entry is to be returned.</td>
</tr>
<tr>
<td><code>BeginMatrix</code></td>
<td>Indicates that the formatting string for beginning a matrix is to be returned.</td>
</tr>
<tr>
<td><code>BeginRow</code></td>
<td>Indicates that the formatting string for beginning a row is to be returned.</td>
</tr>
<tr>
<td><code>BeginRowLabel</code></td>
<td>Indicates that the formatting string for beginning a row label is to be returned.</td>
</tr>
<tr>
<td><code>ColumnLabel</code></td>
<td>Indicates that the formatted string for a given column label is to be returned.</td>
</tr>
<tr>
<td><code>EndColumnLabel</code></td>
<td>Indicates that the formatting string for ending a column label is to be returned.</td>
</tr>
<tr>
<td><code>EndColumnLabels</code></td>
<td></td>
</tr>
</tbody>
</table>
**Description**
Indicates that the formatting string for ending a column label row is to be returned.

**EndEntry**
public Imsl.Math.PrintMatrixFormat.FormatType EndEntry

**Description**
Indicates that the formatted string for ending an entry is to be returned.

**EndMatrix**
public Imsl.Math.PrintMatrixFormat.FormatType EndMatrix

**Description**
Indicates that the formatting string for ending a matrix is to be returned.

**EndRow**
public Imsl.Math.PrintMatrixFormat.FormatType EndRow

**Description**
Indicates that the formatting string for ending a row is to be returned.

**EndRowLabel**
public Imsl.Math.PrintMatrixFormat.FormatType EndRowLabel

**Description**
Indicates that the formatting string for ending a row label is to be returned.

**Entry**
public Imsl.Math.PrintMatrixFormat.FormatType Entry

**Description**
Indicates that the formatted string for a given entry is to be returned.

**RowLabel**
public Imsl.Math.PrintMatrixFormat.FormatType RowLabel

**Description**
Indicates that the formatted string for a given row label is to be returned.

---

**PrintMatrixFormat.ColumnLabelType Enumeration**

public enumeration Imsl.Math.PrintMatrixFormat.ColumnLabelType
Type for column labels.
Fields

**LabelNone**

public `Imsl.Math.PrintMatrixFormat.ColumnLabelType` `LabelNone`

**Description**

Specifies no column labels will be displayed.

**LabelNumber**

public `Imsl.Math.PrintMatrixFormat.ColumnLabelType` `LabelNumber`

**Description**

Specifies column labels will be an array of `int`s.

**LabelString**

public `Imsl.Math.PrintMatrixFormat.ColumnLabelType` `LabelString`

**Description**

Specifies column labels will be an array of `String`s.

---

**PrintMatrixFormat.RowLabelType** Enumeration

public enumeration `Imsl.Math.PrintMatrixFormat.RowLabelType`

Type for row labels.

Fields

**LabelNone**

public `Imsl.Math.PrintMatrixFormat.RowLabelType` `LabelNone`

**Description**

Specifies no row labels will be displayed.

**LabelNumber**

public `Imsl.Math.PrintMatrixFormat.RowLabelType` `LabelNumber`

**Description**

Specifies row labels will be an array of `int`s.
Chapter 12: Basic Statistics

Types

class Summary ................................................................. 534
class Covariances ............................................................... 545
enumeration Covariances.MatrixType ..................................... 551
class PartialCovariances ......................................................... 552
class NormOneSample .......................................................... 559
class NormTwoSample ........................................................... 565
class Sort ............................................................... 576
class Ranks ............................................................ 584
enumeration Ranks.Tie .......................................................... 592
class EmpiricalQuantiles ......................................................... 593
class TableOneWay .............................................................. 596
class TableTwoWay .............................................................. 601
class TableMultiWay ........................................................... 608
class TableMultiWay.TableBalanced .......................................... 614
class TableMultiWay.TableUnbalanced ...................................... 615

Usage Notes

The methods/classes for the computations of basic statistics generally have relatively simple arguments. Most of the methods/classes in this chapter allow for missing values. Missing value codes can be set by using Double.NaN.

Several methods/classes in this chapter perform statistical tests. These methods in the classes generally return a "p-value" for the test. The p-value is between 0 and 1 and is the probability of observing data that would yield a test statistic as extreme or more extreme under the assumption of the null hypothesis. Hence, a small p-value is evidence for the rejection of the null hypothesis.
Summary Class

public class Imsl.Stat.Summary
Computes basic univariate statistics.

For the data in \( x \), \texttt{Summary} computes the sample mean, variance, minimum, maximum, and other basic statistics. It also computes confidence intervals for the mean and variance if the sample is assumed to be from a normal population.

Missing values, that is, values equal to NaN (not a number), are excluded from the computations. The sum of the weights is used only in computing the mean (of course, then the weighted mean is used in computing the central moments). The definitions of some of the statistics are given below in terms of a single variable \( x \). The \( i \)-th datum is \( x_i \), with corresponding weight \( w_i \). If weights are not specified, the \( w_i \) are identically one. The summation in each case is over the set of valid observations, based on the presence of missing values in the data.

Number of nonmissing observations,

\[
n = \sum f_i
\]

Mean,

\[
\bar{x}_w = \frac{\sum f_i w_i x_i}{\sum f_i w_i}
\]

Variance,

\[
s_w^2 = \frac{\sum f_i w_i (x_i - \bar{x}_w)^2}{n - 1}
\]

Skewness,

\[
\frac{\sum f_i w_i (x_i - \bar{x}_w)^3}{(\sum f_i w_i (x_i - \bar{x}_w)^2 / n)^{3/2}}
\]

Excess or Kurtosis,

\[
\frac{\sum f_i w_i (x_i - \bar{x}_w)^4}{(\sum f_i w_i (x_i - \bar{x}_w)^2 / n)^2} - 3
\]
Minimum,
\[ x_{\text{min}} = \min(x_i) \]

Maximum,
\[ x_{\text{max}} = \max(x_i) \]

**Constructor**

**Summary**

```java
public Summary()
```

**Description**

Constructs a new summary statistics object.

**Methods**

**GetConfidenceMean**

```java
public double[] GetConfidenceMean(double p)
```

**Description**

Returns the confidence interval for the mean (assuming normality).

**Parameter**

- `p` – A `double` which specifies the confidence level desired, usually 0.90, 0.95 or 0.99.

**Returns**

A `double` array of length 2 which contains the lower and upper confidence limits for the mean.

**GetConfidenceVariance**

```java
public double[] GetConfidenceVariance(double p)
```

**Description**

Returns the confidence interval for the variance (assuming normality).

**Parameter**

- `p` – A `double` which specifies the confidence level desired, usually 0.90, 0.95 or 0.99.
Returns
A double array of length 2 which contains the lower and upper confidence limits for the variance.

GetKurtosis
public double GetKurtosis()

Description
Returns the kurtosis.

Returns
A double representing the kurtosis.

GetKurtosis
static public double GetKurtosis(double[] x)

Description
Returns the kurtosis of the given data set.

Parameter
x – A double array containing the data set whose kurtosis is to be found.

Returns
A double which specifies the kurtosis of the given data set.

GetKurtosis
static public double GetKurtosis(double[] x, double[] weight)

Description
Returns the kurtosis of the given data set and associated weights.

Parameters
x – A double array containing the data set whose kurtosis is to be found.
weight – A double array containing the weights associated with the data points x.

Returns
A double which specifies the kurtosis of the given data set.

GetMaximum
public double GetMaximum()

Description
Returns the maximum.

Returns
A double representing the maximum.

GetMaximum
static public double GetMaximum(double[] x)
Description
Returns the maximum of the given data set.

Parameter
x – A double array containing the data set whose maximum is to be found.

Returns
A double which specifies the maximum of the given data set.

GetMaximum
static protected internal int GetMaximum(int[] x)

Description
Returns the maximum of the given data set.

Parameter
x – An int array containing the data set whose maximum is to be found.

Returns
An int which specifies the maximum of the given data set.

GetMean
public double GetMean()

Description
Returns the population mean.

Returns
A double representing the population mean.

GetMean
static public double GetMean(double[] x)

Description
Returns the mean of the given data set.

Parameter
x – A double array containing the data set whose mean is to be found.

Returns
A double which specifies the mean of the given data set.

GetMean
static public double GetMean(double[] x, double[] weight)

Description
Returns the mean of the given data set with associated weights.
Parameters

- **x** – A double array containing the data set whose mean is to be found.
- **weight** – A double array containing the weights associated with the data points **x**.

Returns

A double which specifies the mean of the given data set.

GetMedian

```csharp
static public double GetMedian(double[] x)
```

Description

Returns the median of the given data set.

Parameter

- **x** – A double array containing the data set whose median is to be found.

Returns

A double which specifies the median of the given data set.

GetMinimum

```csharp
public double GetMinimum()
```

Description

Returns the minimum.

Returns

A double representing the minimum.

GetMinimum

```csharp
static public double GetMinimum(double[] x)
```

Description

Returns the minimum of the given data set.

Parameter

- **x** – A double array containing the data set whose minimum is to be found.

Returns

A double which specifies the minimum of the given data set.

GetMinimum

```csharp
static protected internal int GetMinimum(int[] x)
```

Description

Returns the minimum of the given data set.

Parameter

- **x** – An int array containing the data set whose minimum is to be found.
Returns
An int which specifies the minimum of the given data set.

GetMode
static public double GetMode(double[] x)

Description
Returns the mode of the given data set.

Parameter
x – A double array containing the data set whose mode is to be found.

Returns
A double which specifies the mode of the given data set.

Remarks
Ties are broken at random.

GetSampleStandardDeviation
public double GetSampleStandardDeviation()

Description
Returns the sample standard deviation.

Returns
A double representing the sample standard deviation.

GetSampleStandardDeviation
static public double GetSampleStandardDeviation(double[] x)

Description
Returns the sample standard deviation of the given data set.

Parameter
x – A double array containing the data set whose sample standard deviation is to be found.

Returns
A double which specifies the sample standard deviation of the given data set.

GetSampleStandardDeviation
static public double GetSampleStandardDeviation(double[] x, double[] weight)

Description
Returns the sample standard deviation of the given data set and associated weights.

Parameters
x – A double array containing the data set whose sample standard deviation is to be found.
weight – A double array containing the weights associated with the data points x.
Returns
A double which specifies the sample standard deviation of the given data set.

GetSampleVariance
public double GetSampleVariance()

Description
Returns the sample variance.

Returns
A double representing the sample variance.

GetSampleVariance
static public double GetSampleVariance(double[] x)

Description
Returns the sample variance of the given data set.

Parameter
x – A double array containing the data set whose sample variance is to be found.

Returns
A double which specifies the sample variance of the given data set.

GetSampleVariance
static public double GetSampleVariance(double[] x, double[] weight)

Description
Returns the sample variance of the given data set and associated weights.

Parameters
x – A double array containing the data set whose sample variance is to be found.
weight – A double array containing the weights associated with the data points x.

Returns
A double which specifies the sample variance of the given data set.

GetSkewness
public double GetSkewness()

Description
Returns the skewness.

Returns
A double representing the skewness.

GetSkewness
static public double GetSkewness(double[] x)
Description
Returns the skewness of the given data set.

Parameter
x – A double array containing the data set whose skewness is to be found.

Returns
A double which specifies the skewness of the given data set.

GetSkewness
static public double GetSkewness(double[] x, double[] weight)

Description
Returns the skewness of the given data set and associated weights.

Parameters
x – A double array containing the data set whose skewness is to be found.
weight – A double array containing the weights associated with the data points x.

Returns
A double which specifies the skewness of the given data set.

GetStandardDeviation
public double GetStandardDeviation()

Description
Returns the population standard deviation.

Returns
A double representing the population standard deviation.

GetStandardDeviation
static public double GetStandardDeviation(double[] x)

Description
Returns the population standard deviation of the given data set.

Parameter
x – A double array containing the data set whose standard deviation is to be found.

Returns
A double which specifies the population standard deviation of the given data set.

GetStandardDeviation
static public double GetStandardDeviation(double[] x, double[] weight)

Description
Returns the population standard deviation of the given data set and associated weights.

Basic Statistics
Parameters
- x – A double array containing the data set whose standard deviation is to be found.
- weight – A double array containing the weights associated with the data points x.

Returns
A double which specifies the population standard deviation of the given data set.

GetVariance
public double GetVariance()

Description
Returns the population variance.

Returns
A double representing the population variance.

GetVariance
static public double GetVariance(double[] x)

Description
Returns the population variance of the given data set.

Parameter
- x – A double array containing the data set whose population variance is to be found.

Returns
A double which specifies the population variance of the given data set.

GetVariance
static public double GetVariance(double[] x, double[] weight)

Description
Returns the population variance of the given data set and associated weights.

Parameters
- x – A double array containing the data set whose population variance is to be found.
- weight – A double array containing the weights associated with the data points x.

Returns
A double which specifies the population variance of the given data set.

Update
public void Update(double x)

Description
Adds an observation to the Summary object.
Parameter

- A double which specifies the data observation to be added.

Update

public void Update(double x, double weight)

Description

Adds an observation and associated weight to the Summary object.

Parameters

- x – A double which specifies the data observation to be added.
- weight – A double which specifies the weight associated with the observation.

Update

public void Update(double[] x)

Description

Adds a set of observations to the Summary object.

Parameter

- x – A double array of data observations to be added.

Update

public void Update(double[] x, double[] weight)

Description

Adds a set of observations and associated weights to the Summary object.

Parameters

- x – A double array of data observations to be added.
- weight – A double array of weights associated with the observations.

Example: Summary Statistics

Summary statistics for a small data set are computed.

```csharp
using System;
using Imsl.Stat;

public class SummaryEx1
{
    internal static readonly double[] data1 =
        new double[] { 3, 6.4, 2, 1.6, -8, 12,
                      -7, 6.4, 22, 1, 0, -3.2};

    public static void Main(String[] args)
    {
        Summary summary = new Summary();
        summary.Update(data1);
    }
}
```

Basic Statistics

Summary • 543
Console.Out.WriteLine("The minimum is " + summary.GetMinimum());
Console.Out.WriteLine();

Console.Out.WriteLine("The maximum is " + summary.GetMaximum());
Console.Out.WriteLine();

Console.Out.WriteLine("The mean is " + summary.GetMean());
Console.Out.WriteLine();

Console.Out.WriteLine("The variance is " + summary.GetVariance());
Console.Out.WriteLine();

Console.Out.WriteLine("The sample variance is " + summary.GetSampleVariance());
Console.Out.WriteLine();

Console.Out.WriteLine("The standard deviation is " + summary.GetStandardDeviation());
Console.Out.WriteLine();

Console.Out.WriteLine("The skewness is " + summary.GetSkewness());
Console.Out.WriteLine();

Console.Out.WriteLine("The kurtosis is " + summary.GetKurtosis());
Console.Out.WriteLine();

double[] confmn = new double[2];
confmn = summary.GetConfidenceMean(0.95);
Console.Out.WriteLine("The confidence Mean is {" + confmn[0] + ", " + confmn[1] + "}");
Console.Out.WriteLine();

double[] confvr = new double[2];
confvr = summary.GetConfidenceVariance(0.95);
Console.Out.WriteLine("The confidence Variance is {" + confvr[0] + ", " + confvr[1] + "}");

}]

Output
The minimum is -8
The maximum is 22
The mean is 3.01666666666667
The variance is 61.70972222222222
The sample variance is 67.319696969697
The standard deviation is 7.85555359107315
The skewness is 0.863222413428583
The kurtosis is 0.567706048385121
The confidence Mean is {-2.19645146860124, 8.22978480193457}
The confidence Variance is {33.7826187272066, 194.068533277244}

Covariances Class

public class Imsl.Stat.Covariances

Computes the sample variance-covariance or correlation matrix.

Class Covariances computes estimates of correlations, covariances, or sums of squares and crossproducts for a data matrix \( x \). Weights and frequencies are allowed but not required.

The means, (corrected) sums of squares, and (corrected) sums of crossproducts are computed using the method of provisional means. Let \( x_{ki} \) denote the mean based on \( i \) observations for the \( k \)-th variable, \( f_i \) denote the frequency of the \( i \)-th observation, \( w_i \) denote the weight of the \( i \)-th observations, and \( c_{jki} \) denote the sum of crossproducts (or sum of squares if \( j = k \)) based on \( i \) observations. Then the method of provisional means finds new means and sums of crossproducts as shown in the example below.

The means and crossproducts are initialized as follows:

\[
x_{k0} = 0.0 \quad \text{for } k = 1, \ldots, p
\]

\[
c_{jk0} = 0.0 \quad \text{for } j, k = 1, \ldots, p
\]

where \( p \) denotes the number of variables. Letting \( x_{k,i+1} \) denote the \( k \)-th variable of observation \( i + 1 \), each new observation leads to the following updates for \( x_{ki} \) and \( c_{jki} \) using the update constant \( r_{i+1} \):

\[
r_{i+1} = \frac{f_{i+1} w_{i+1}}{\sum_{l=1}^{i+1} f_l w_l}
\]

\[
\tilde{x}_{k,i+1} = \tilde{x}_{ki} + (x_{k,i+1} - \tilde{x}_{ki}) r_{i+1}
\]
\[ c_{jk,i+1} = c_{jk,i} + f_{i+1} \hat{w}_{i+1} \left( x_{j,i+1} - \bar{x}_{ji} \right) \left( x_{k,i+1} - \bar{x}_{ki} \right) \left( 1 - r_{i+1} \right) \]

The default value for weights and frequencies is 1. Means and variances are computed based on the valid data for each variable or, if required, based on all the valid data for each pair of variables.

**Properties**

**MissingValueMethod**

public int MissingValueMethod {get; set; }

**Description**

Sets the method used to exclude missing values in \( x \) from the computations.

**Property Value**

An int scalar indicating the method to use.

**Remarks**

The methods are as follows:

<table>
<thead>
<tr>
<th>MissingValueMethod</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The exclusion is listwise, default. (The entire row of ( x ) is excluded if any of the values of the row is equal to the missing value code.)</td>
</tr>
<tr>
<td>1</td>
<td>Raw crossproducts are computed from all valid pairs and means, and variances are computed from all valid data on the individual variables. Corrected crossproducts, covariances, and correlations are computed using these quantities.</td>
</tr>
<tr>
<td>2</td>
<td>Raw crossproducts, means, and variances are computed as in the case of MissingValueMethod = 1. However, corrected crossproducts and covariances are computed only from the valid pairs of data. Correlations are computed using these covariances and the variances from all valid data.</td>
</tr>
<tr>
<td>3</td>
<td>Raw crossproducts, means, variances, and covariances are computed as in the case of MissingValueMethod = 2. Correlations are computed using these covariances, but the variances used are computed from the valid pairs of data.</td>
</tr>
</tbody>
</table>

Double.NaN is interpreted as the missing value code.

**NumberOfProcessors**

public int NumberOfProcessors {get; set; }

---

546 • Covariances IMLS C# Numerical Library
Description
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

Property Value
An int indicating the maximum possible number of processors to use.

Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

NumRowMissing
public int NumRowMissing {get; }

Description
Returns the total number of observations that contain any missing values (Double.NaN). Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is 0.

Property Value
An int scalar containing the total number of observations that contain any missing values (Double.NaN).

Observations
public int Observations {get; }

Description
Returns the sum of the frequencies. Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is 0.

Property Value
An int scalar containing the sum of the frequencies.

Remarks
If MissingValueMethod = 0, observations with missing values are not included. Otherwise, all observations are included except for observations with missing values for the weight or the frequency.

SumOfWeights
public double SumOfWeights {get; }

Description
Returns the sum of the weights of all observations. Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is 0.

Property Value
A double scalar containing the sum of the weights of all observations.
Remarks
If MissingValueMethod = 0, observations with missing values are not included. Otherwise, all observations are included except for observations with missing values for the weight or the frequency.

Constructor

Covariances
public Covariances(double[,] x)

Description
Constructor for Covariances.

Parameter
x – A double matrix containing the data.

Exception
System.ArgumentException is thrown if x.GetLength(0), and x.GetLength(1) are equal to 0.

Methods

Compute
public double[,] Compute(Imsl.Stat.Covariances.MatrixType matrixType)

Description
Computes the matrix.

Parameter
matrixType – A Covariances.MatrixType indicating the type of matrix to compute.

Returns
A double matrix containing computed result.

Exceptions
Imsl.Stat.MoreObsDelThanEnteredException is thrown if more observations are being deleted from “variance-covariance” matrix than were originally entered.
The corresponding row, column of the incidence matrix is less than zero.
Imsl.Stat.DiffObsDeletedException is thrown if different observations are being deleted than were originally entered.

GetIncidenceMatrix
public int[,] GetIncidenceMatrix()
Returns the incidence matrix. Note that the `Compute` method must be invoked first before invoking this method. Otherwise, the method throws a `NullReferenceException` exception.

**Returns**

An `int` matrix containing the incidence matrix.

**Remarks**

If `MissingValueMethod` is 0, incidence matrix is 1 x 1 and contains the number of valid observations; otherwise, incidence matrix is `x.GetLength(1)` x `x.GetLength(1)` and contains the number of pairs of valid observations used in calculating the crossproducts for covariance.

---

### GetMeans

```csharp
public double[] GetMeans()
```

**Description**

Returns the means of the variables in `x`.

**Returns**

A `double` array containing the means of the variables in `x`.

**Remarks**

The components of the array correspond to the columns of `x`. Note that the `Compute` method must be invoked first before invoking this method. Otherwise, the method throws a `NullReferenceException` exception.

---

### SetFrequencies

```csharp
public void SetFrequencies(double[] frequencies)
```

**Description**

The frequency for each observation.

**Parameter**

- `frequencies` – A `double` array of size `x.GetLength(0)` containing the frequency for each observation.

**Remarks**

By default, `frequencies[] = 1`.

---

### SetWeights

```csharp
public void SetWeights(double[] weights)
```

**Description**

Sets the weight for each observation.

**Parameter**

- `weights` – A `double` array of size `x.GetLength(0)` containing the weight for each observation.

---

Basic Statistics | Covariances • 549
Remarks
By default, weights[] = 1.

Example: Covariances
This example illustrates the use of Covariances class for the first 50 observations in the Fisher iris data (Fisher 1936). Note that the first variable is constant over the first 50 observations.

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;

public class CovariancesEx1
{
    public static void Main(String[] args)
    {
        double[,] x = {{1.0, 5.1, 3.5, 1.4, .2},
                        {1.0, 4.9, 3.0, 1.4, .2},
                        {1.0, 4.7, 3.2, 1.3, .2},
                        {1.0, 4.6, 3.1, 1.5, .2},
                        {1.0, 5.0, 3.6, 1.4, .2},
                        {1.0, 5.4, 3.9, 1.7, .4},
                        {1.0, 4.6, 3.4, 1.4, .3},
                        {1.0, 5.0, 3.4, 1.5, .2},
                        {1.0, 4.4, 2.9, 1.4, .2},
                        {1.0, 4.9, 3.1, 1.5, .1},
                        {1.0, 5.4, 3.7, 1.5, .2},
                        {1.0, 4.8, 3.4, 1.6, .2},
                        {1.0, 4.8, 3.0, 1.4, .1},
                        {1.0, 4.3, 3.0, 1.1, .1},
                        {1.0, 5.8, 4.0, 1.2, .2},
                        {1.0, 5.7, 4.4, 1.5, .4},
                        {1.0, 5.4, 3.9, 1.3, .4},
                        {1.0, 5.1, 3.5, 1.4, .3},
                        {1.0, 5.7, 3.8, 1.7, .3},
                        {1.0, 5.1, 3.8, 1.5, .3},
                        {1.0, 5.4, 3.4, 1.7, .2},
                        {1.0, 5.1, 3.7, 1.5, .4},
                        {1.0, 4.6, 3.6, 1.0, .2},
                        {1.0, 5.1, 3.3, 1.7, .5},
                        {1.0, 4.8, 3.4, 1.9, .2},
                        {1.0, 5.0, 3.0, 1.6, .2},
                        {1.0, 5.0, 3.4, 1.6, .4},
                        {1.0, 5.2, 3.5, 1.5, .2},
                        {1.0, 5.2, 3.4, 1.4, .2},
                        {1.0, 4.7, 3.2, 1.6, .2},
                        {1.0, 4.8, 3.1, 1.6, .2},
                        {1.0, 5.4, 3.4, 1.5, .4},
                        {1.0, 5.2, 4.1, 1.5, .1},
                        {1.0, 5.5, 4.2, 1.4, .2},
                        {1.0, 4.9, 3.1, 1.5, .2},
                        {1.0, 5.0, 3.2, 1.2, .2},
                        {1.0, 5.5, 3.5, 1.3, .2}};
```
```java
Covariances co = new Covariances(x);

PrintMatrix pm =
    new PrintMatrix("Sample Variances-covariances Matrix");
PrintMatrixFormat pmf = new PrintMatrixFormat();
    pmf.NumberFormat = "0.0000";
    pm.SetMatrixType(PrintMatrix.MatrixType.UpperTriangular);

    pm.Print(pmf,
        co.Compute(Covariances.MatrixType.VarianceCovariance));

Output
Sample Variances-covariances Matrix
0 1 2 3 4
0 0.0000 0.0000 0.0000 0.0000 0.0000
1 0.1242 0.0992 0.0164 0.0103
2 0.1437 0.0117 0.0093
3 0.0302 0.0061
4 0.0111
```

### Covariances.MatrixType Enumeration

```java
public enum Covariances.MatrixType {
    BasicStatistics,
    VarianceCovariance
}
```

**Basic Statistics Covariances.MatrixType**

- Specifies the type of matrix to be computed.
Fields

**CorrectedSSCP**

public Imsl.Stat.Covariances.MatrixType CorrectedSSCP

**Description**

Indicates corrected sums of squares and crossproducts matrix.

**Correlation**

public Imsl.Stat.Covariances.MatrixType Correlation

**Description**

Indicates correlation matrix.

**StdevCorrelation**

public Imsl.Stat.Covariances.MatrixType StdevCorrelation

**Description**

Indicates correlation matrix except for the diagonal elements which are the standard deviations.

**VarianceCovariance**

public Imsl.Stat.Covariances.MatrixType VarianceCovariance

**Description**

Indicates variance-covariance matrix.

---

**PartialCovariances Class**

public class Imsl.Stat.PartialCovariances

Class PartialCovariances computes the partial covariances or partial correlations from an input covariance or correlation matrix.

If the “independent” variables (the linear “effect” of the independent variables is removed in computing the partial covariances/correlations) are linearly related to one another, PartialCovariances detects the linearity and eliminates one or more of the independent variables from the list of independent variables. The number of variables eliminated, if any, can be determined from the property PartialDegreesOfFreedom.

Given a covariance or correlation matrix $\Sigma$ partitioned as

$$
\begin{pmatrix}
\Sigma_{11} & \Sigma_{12} \\
\Sigma_{21} & \Sigma_{22}
\end{pmatrix}
$$
class PartialCovariances computes the partial covariances (of the standardized variables if $\Sigma$ is a correlation matrix) as
\[
\Sigma_{22|1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}
\]
A positive semidefinite solver is used to compute $\Sigma_{11}^{-1}\Sigma_{12}$.
If partial correlations are desired, these are computed as
\[
P_{22|1} = \left[\text{diag}(\Sigma_{22|1})\right]^{-1/2}\Sigma_{22|1}\left[\text{diag}(\Sigma_{22|1})\right]^{-1/2}
\]
where $\text{diag}(\Sigma)$ denotes the matrix containing the diagonal of its argument along its diagonal with zeros off the diagonal. If $\Sigma_{11}$ is singular, then as many variables as required are deleted from $\Sigma_{11}$ (and $\Sigma_{12}$) in order to eliminate the linear dependencies. The computations then proceed as above.

The $p$-value for a partial covariance tests the null hypothesis $H_0: \sigma_{ij|1} = 0$, where $\sigma_{ij|1}$ is the $(i,j)$ element in matrix $\Sigma_{22|1}$. The $p$-value for a partial correlation tests the null hypothesis $H_0: \rho_{ij|1} = 0$, where $\rho_{ij|1}$ is the $(i,j)$ element in matrix $P_{22|1}$. The $p$-values are returned by GetPValues. If the degrees of freedom for $\sigma_{ij|1}$, $df$, is not known, the resulting $p$-values may be useful for comparison, but they should not be used as an approximation to the actual probabilities.

**Properties**

**NumberOfProcessors**
public int NumberOfProcessors {get; set; }

**Description**
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**
An int indicating the maximum possible number of processors to use.

**Remarks**
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**PartialDegreesOfFreedom**
public int PartialDegreesOfFreedom {get; }

**Description**
The degrees of freedom in the test that the partial correlation (covariance) is zero.
Remarks
This will usually be df minus the rank of the independent variables, but will be greater than this value if the independent variables are computationally linearly related. If this value is not greater than one then there are not enough degrees of freedom for hypothesis testing. A warning is also issued in this case.

Constructors

PartialCovariances

public PartialCovariances(int nIndependent, double[,] sigma, int df)

Description
Creates a PartialCovariances object from a covariance or correlation matrix with a mix of dependent and independent variables.

Parameters
- nIndependent – is the number of “independent” variables to be used in the partial covariances/correlations. The partial covariances/correlations are the covariances/correlations between the dependent variables after removing the linear effect of the independent variables.
- sigma – is a correlation or covariance matrix. The rows/columns must be ordered such that the first nIndependent rows/columns contain the independent variables, followed by the row/columns containing the dependent variables. The matrix must always be symmetric, positive semidefinite.
- df – is an int indicating the number of degrees of freedom associated with the input matrix. If the number of degrees of freedom in the matrix varies from element to element, then a conservative choice for df is the minimum degrees of freedom for all elements in the matrix. The value of df must be at least one.

Exceptions
- Imsl.Stat.InvalidMatrixException is thrown if a computed correlation is greater than one for some pair of variables.
- Imsl.Stat.InvalidPartialCorrelationException is thrown if a computed partial correlation is greater than one for some pair of variables. The input matrix to the constructor was not positive semidefinite.

PartialCovariances

public PartialCovariances(int[] xIndices, double[,] sigma, int df)

Description
Creates a PartialCovariances object from a covariance or correlation matrix with a mix of dependent and independent variables.
Parameters

xIndices – is an array containing values indicating the status of the variable. If the \( i \)-th entry is 0 then the \( i \)-th column of the matrix contains a dependent variable. If the \( i \)-th entry is positive then the \( i \)-th column of the matrix contains an independent variable. If the \( i \)-th entry is negative then the \( i \)-th column of the matrix is not used in the analysis.

\( \sigma \) – is a correlation or covariance matrix. The number of rows and columns in \( \sigma \) must equal the length of the array xIndices. The matrix must always be symmetric, positive semidefinite.

df – is an int indicating the number of degrees of freedom associated with the input matrix. If the number of degrees of freedom in the matrix varies from element to element, then a conservative choice for df is the minimum degrees of freedom for all elements in the matrix. The value of df must be at least one.

Exceptions

Imsl.Stat.InvalidMatrixException is thrown if a computed correlation is greater than one for some pair of variables.

Imsl.Stat.InvalidPartialCorrelationException is thrown if a computed partial correlation is greater than one for some pair of variables. The input matrix to the constructor was not positive semidefinite.

Methods

GetPartialCorrelationMatrix
public double[,] GetPartialCorrelationMatrix()

Description
Returns the partial correlation matrix.

Returns
The partial correlation matrix.

Remarks
This is valid only if the input to the constructor was a correlation matrix.

GetPartialCovarianceMatrix
public double[,] GetPartialCovarianceMatrix()

Description
Returns the partial covariance matrix.

Returns
The partial covariance matrix.
Remarks
This is valid only if the input to the constructor was a covariance matrix.

GetPValues
public double[,] GetPValues()

Description
Calculates the p-values for testing the null hypothesis that the associated partial covariance/correlation is zero.

Returns
A square array of type double containing the p-values. The order of the matrix equals the number of dependent variables.

If the partial degrees of freedom is not greater than one then there are not enough degrees of freedom for hypothesis testing. The returned matrix will be set to all NaN values in this case. A warning is also issued in this case.

Remarks
It is assumed that the observations from which sigma was computed follows a multivariate normal distribution and that each element in sigma has df degrees of freedom.

Example 1
This example computes partial covariances, scaled from a nine-variable correlation matrix originally given by Emmett (1949). The first three rows and columns contain the independent variables and the final six rows and columns contain the dependent variables.

using System;
using PartialCovariances = Imsl.Stat.PartialCovariances;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;

public class PartialCovariancesEx1
{
    static public void Main(String[] arg)
    {
        double[,] sigma =
        {
            {6.300, 3.050, 1.933, 3.365, 1.317, 2.293, 2.586, 1.242, 4.363},
            {3.050, 5.400, 2.170, 3.346, 1.473, 2.303, 2.274, 0.750, 4.077},
            {1.933, 2.170, 3.800, 1.970, 0.798, 1.062, 1.576, 0.487, 2.673},
            {3.365, 3.346, 1.970, 8.100, 2.983, 4.828, 2.255, 0.925, 3.910},
            {1.317, 1.473, 0.798, 2.983, 2.300, 2.209, 1.039, 0.258, 1.687},
            {2.293, 2.303, 1.062, 4.828, 2.209, 4.600, 1.427, 0.768, 2.754},
            {2.586, 2.274, 1.576, 2.255, 1.039, 1.427, 3.200, 0.785, 3.309},
            {1.242, 0.750, 0.487, 0.925, 0.258, 0.768, 0.785, 1.300, 1.458},
            {4.363, 4.077, 2.673, 3.910, 1.687, 2.754, 3.309, 1.458, 7.400}
        };
        int nIndependent = 3;
        int df = 30;
PartialCovariances pcov = new PartialCovariances(nIndependent, sigma, df);

double[,] covar = pcov.GetPartialCovarianceMatrix();
PrintMatrixFormat pmf = new PrintMatrixFormat();
pmf.NumberFormat = "0.0000";
new PrintMatrix("Partial Covariances").Print(pmf,covar);

int pdf = pcov.PartialDegreesOfFreedom;
Console.Out.WriteLine("Partial Degrees of Freedom " + pdf);
Console.Out.WriteLine();

double[,] pvalues = pcov.GetPValues();
new PrintMatrix("p Values").Print(pmf,pvalues);
}

Output

Partial Covariances

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>1</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>3</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>5.4953</td>
<td>1.8955</td>
<td>3.0836</td>
</tr>
<tr>
<td>4</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>1.8955</td>
<td>1.8407</td>
<td>1.4764</td>
</tr>
<tr>
<td>5</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>3.0836</td>
<td>1.4764</td>
<td>3.4026</td>
</tr>
</tbody>
</table>

Partial Degrees of Freedom 27

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>1</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>3</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0008</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>4</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0008</td>
<td>0.0000</td>
<td>0.0009</td>
</tr>
<tr>
<td>5</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0009</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Example 2

This example computes partial correlations from a 9 variable correlation matrix originally given by Emmett (1949). The partial correlations between the remaining variables, after adjusting for variables 1, 3 and 9, are computed. Note in the output that the row and column labels are numbers, not variable numbers. The corresponding variable numbers would be 2, 4, 5, 6, 7 and 8, respectively.
```csharp
static public void Main(String[] arg)
{
    double[,] sigma =
    {
        { 1.000, 0.523, 0.395, 0.471, 0.346, 0.426, 0.576, 0.434, 0.639 },
        { 0.523, 1.000, 0.479, 0.506, 0.418, 0.462, 0.547, 0.283, 0.645 },
        { 0.395, 0.479, 1.000, 0.355, 0.270, 0.254, 0.452, 0.219, 0.504 },
        { 0.471, 0.506, 0.355, 1.000, 0.691, 0.791, 0.443, 0.285, 0.505 },
        { 0.346, 0.418, 0.270, 0.691, 1.000, 0.679, 0.383, 0.149, 0.409 },
        { 0.426, 0.462, 0.254, 0.791, 0.679, 1.000, 0.372, 0.314, 0.472 },
        { 0.576, 0.547, 0.452, 0.443, 0.383, 0.372, 1.000, 0.385, 0.680 },
        { 0.434, 0.283, 0.219, 0.285, 0.149, 0.314, 0.385, 1.000, 0.470 },
        { 0.639, 0.645, 0.504, 0.505, 0.409, 0.472, 0.680, 0.470, 1.000 }
    };

    int[] xIndices = { 1, 0, 1, 0, 0, 0, 0, 0, 1 };
    int df = 30;

    PartialCovariances pcov = new PartialCovariances(xIndices, sigma, df);

    double[,] correl = pcov.GetPartialCorrelationMatrix();
    PrintMatrixFormat pmf = new PrintMatrixFormat();
    pmf.NumberFormat = "0.0000";
    new PrintMatrix("Partial Correlations").Print(pmf, correl);

    double[,] pValues = pcov.GetPValues();
    new PrintMatrix("P-Values").Print(pmf, pValues);
}
```

Output

<table>
<thead>
<tr>
<th>Partial Correlations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P-Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
NormOneSample Class

public class Imsl.Stat.NormOneSample

Computes statistics for mean and variance inferences using a sample from a normal population.

The statistics for mean and variance inferences are computed by using a sample from a normal population, including methods for the confidence intervals and tests for both mean and variance. The definitions of mean and variance are given below. The summation in each case is over the set of valid observations, based on the presence of missing values in the data.

Property Mean, returns value

\[ \bar{x} = \frac{\sum x_i}{n} \]

\[ \Delta^d Z_t \]

Property StdDev, returns value

\[ s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}} \]

The property TTest returns the \( t \) statistic for the two-sided test concerning the population mean which is given by

\[ t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \]

where \( s \) and \( \bar{x} \) are given above. This quantity has a \( T \) distribution with \( n - 1 \) degrees of freedom. The property TTestDF returns the degree of freedom.

Property ChiSquaredTest returns the chi-squared statistic for the two-sided test concerning the population variance which is given by

\[ \chi^2 = \frac{(n - 1)s^2}{\sigma^2_0} \]

where \( s \) is given above. This quantity has a \( \chi^2 \) distribution with \( n - 1 \) degrees of freedom. Property ChiSquaredTestDF returns the degrees of freedom.
Properties

ChiSquaredTest
public double ChiSquaredTest {get; }

Description
Returns the test statistic associated with the chi-squared test for variances.

Property Value
A double containing the test statistic for the chi-squared test.

Remarks
The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where $\omega_0^2$ is the null hypothesis value as described in ChiSquaredTestNull.

ChiSquaredTestDF
public int ChiSquaredTestDF {get; }

Description
Returns the degrees of freedom associated with the chi-squared test for variances.

Property Value
A int containing the degrees of freedom for the chi-squared test.

Remarks
The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where $\omega_0^2$ is the null hypothesis value as described in ChiSquaredTestNull.

ChiSquaredTestNull
public double ChiSquaredTestNull {get; set; }

Description
The null hypothesis value for the chi-squared test.

Property Value
A double containing the null hypothesis value for the chi-squared test.

Remarks
The default is 1.0.

ChiSquaredTestP
public double ChiSquaredTestP {get; }

Description
Returns the probability of a larger chi-squared associated with the chi-squared test for variances.

Property Value
A double containing the probability of a larger chi-squared for the chi-squared test for variances.
Remarks

The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where $\omega_0^2$ is the null hypothesis value as described in ChiSquaredTestNull.

ConfidenceMean

public double ConfidenceMean {get; set; }

Description

The confidence level (in percent) for a two-sided interval estimate of the mean.

Property Value

A double containing the confidence level of the mean.

Remarks

ConfidenceMean must be between 0.0 and 1.0 and is often 0.90, 0.95 or 0.99. For a one-sided confidence interval with confidence level c less than 50 percent, set ConfidenceMean = 1.0 - 2.0*(c/100). This effectively gives the one-sided confidence interval for both c% and (100-c)% of the distribution is greater than UpperCIMean. If the confidence mean is not specified, a 95-percent confidence interval is computed.

ConfidenceVariance

public double ConfidenceVariance {get; set; }

Description

The confidence level (in percent) for two-sided interval estimate of the variances.

Property Value

A double containing the confidence level of the variance.

Remarks

ConfidenceVariance must be between 0.0 and 1.0 and is often 0.90, 0.95 or 0.99. For a one-sided confidence interval with confidence level c (at least 50 percent), set ConfidenceVariance = 1.0 - 2.0 * (1.0 - c). If the confidence mean is not specified, a 95-percent confidence interval is computed.

LowerCIMean

public double LowerCIMean {get; }

Description

Returns the lower confidence limit for the mean.

Property Value

A double containing the lower confidence limit for the mean.

LowerCIVariance

public double LowerCIVariance {get; }
**Description**
Returns the lower confidence limits for the variance.

**Property Value**
A double containing the lower confidence limits for the variance.

**Mean**
public double Mean {get; }

**Description**
Returns the mean of the sample.

**Property Value**
A double containing the mean of x.

**StdDev**
public double StdDev {get; }

**Description**
Returns the standard deviation of the sample.

**Property Value**
A double containing the standard deviation of the sample.

**TTest**
public double TTest {get; }

**Description**
Returns the test statistic associated with the \( t \) test.

**Property Value**
A double containing the test statistic for the \( t \) test.

**Remarks**
The \( t \) test is a test, against a two-sided alternative, of the null hypothesis value described in TTestNull.

**TTestDF**
public int TTestDF {get; }

**Description**
Returns the degrees of freedom associated with the \( t \) test for the mean.

**Property Value**
A int containing the degrees of freedom for the \( t \) test.

**Remarks**
The \( t \) test is a test, against a two-sided alternative, of the null hypothesis value described in TTestNull.
**Description**
Sets the Null hypothesis value for $t$ test for the mean.

**Property Value**
A double containing the hypothesis value.

**Remarks**
$T_{TestNull} = 0.0$ by default.

---

**TTestP**
public double TTestP {get; }

**Description**
Returns the probability associated with the $t$ test of a larger $t$ in absolute value.

**Property Value**
A double containing the probability for the $t$ test.

**Remarks**
The $t$ test is a test, against a two-sided alternative, of the null hypothesis value described in $T_{TestNull}$.

---

**UpperCIMean**
public double UpperCIMean {get; }

**Description**
Returns the upper confidence limit for the mean.

**Property Value**
A double containing the upper confidence limit for the mean.

---

**UpperCIVariance**
public double UpperCIvariance {get; }

**Description**
Returns the upper confidence limits for the variance.

**Property Value**
A double the upper confidence limits for the variance.

---

**Constructor**

**NormOneSample**
public NormOneSample(double[] x)

**Description**
Constructor to compute statistics for mean and variance inferences using a sample from a normal population.

---

**Basic Statistics**

---

**NormOneSample • 563**
Parameter

\(x\) – A one-dimension double array containing the observations.

Example 1: NormOneSample

This example uses data from Devore (1982, p335), which is based on data published in the *Journal of Materials*. There are 15 observations. The hypothesis \(H_0 : \mu = 20.0\) is tested. The extremely large \(t\) value and the correspondingly small \(p\)-value provide strong evidence to reject the null hypothesis.

```csharp
using System;
using Imsl.Stat;

public class NormOneSampleEx1
{
    public static void Main(String[] args)
    {
        double mean, stdev, lomean, upmean;
        int df;
        double t, pvalue;
        double[] x = new double[] { 26.7, 25.8, 24.0, 24.9, 26.4,
                                   25.9, 24.4, 21.7, 24.1, 25.9,
                                   27.3, 26.9, 27.3, 24.8, 23.6};

        /* Perform Analysis*/
        NormOneSample n1samp = new NormOneSample(x);
        mean = n1samp.Mean;
        stdev = n1samp.StdDev;
        lomean = n1samp.LowerCIMean;
        upmean = n1samp.UpperCIMean;
        n1samp.TTestNull = 20.0;
        df = n1samp.TTestDF;
        t = n1samp.TTest;
        pvalue = n1samp.TTestP;

        /* Print results */
        Console.Out.WriteLine("Sample Mean = " + mean);
        Console.Out.WriteLine("Sample Standard Deviation = " + stdev);
        Console.Out.WriteLine("(95% CI for the mean is " + lomean + " + upmean);
        Console.Out.WriteLine("T Test results");
        Console.Out.WriteLine("df = " + df);
        Console.Out.WriteLine("t = " + t);
        Console.Out.WriteLine("pvalue = " + pvalue);
        Console.Out.WriteLine("\n");

        /* CI variance */
        double ciLoVar = n1samp.LowerCIVariance;
        double ciUpVar = n1samp.UpperCIVariance;
        Console.Out.WriteLine;
    }
}
```

564 • NormOneSample

**IMSL C# Numerical Library**
Output

Sample Mean = 25.3133333333333
Sample Standard Deviation = 1.57881812336528
95% CI for the mean is 24.4390129997097 26.187653666957
T Test results
df = 14
t = 13.0340861992294
pvalue = 3.2147173398634E-09
CI variance is 1.33609260499922 6.19986346723949
Chi-squared Test results
Chi-squared df = 14
Chi-squared t = 34.8973333333333
Chi-squared pvalue = 0.00152231761418218

NormTwoSample Class

public class Imsl.Stat.NormTwoSample

Computes statistics for mean and variance inferences using samples from two normal populations.

Class NormTwoSample computes statistics for making inferences about the means and variances of two normal populations, using independent samples in x1 and x2. For inferences concerning parameters of a single normal population, see class NormOneSample.

Let \( \mu_1 \) and \( \sigma_1^2 \) be the mean and variance of the first population, and let \( \mu_2 \) and \( \sigma_2^2 \) be the corresponding quantities of the second population. The function contains test confidence intervals for difference in means, equality of variances, and the pooled variance.

The means and variances for the two samples are as follows:

\[
\bar{x}_1 = \left( \sum x_{1i} / n_1 \right), \quad \bar{x}_2 = \left( \sum x_{2i} / n_2 \right)
\]

and
\[
s_1^2 = \sum (x_{1i} - \bar{x}_1)^2 / (n_1 - 1), \quad s_2^2 = \sum (x_{2i} - \bar{x}_2)^2 / (n_2 - 1)
\]

**Inferences about the Means**

The test that the difference in means equals a certain value, for example, \(\mu_0\), depends on whether or not the variances of the two populations can be considered equal. If the variances are equal and meanHypothesis equals 0, the test is the two-sample \(t\)-test, which is equivalent to an analysis-of-variance test. The pooled variance for the difference-in-means test is as follows:

\[
s^2 = \frac{(n_1 - 1)s_1 + (n_2 - 1)s_2}{n_1 + n_2 - 2}
\]

The \(t\) statistic is as follows:

\[
t = \frac{\bar{x}_1 - \bar{x}_2 - \mu_0}{s_d \sqrt{1/n_1 + 1/n_2}}
\]

Also, the confidence interval for the difference in means can be obtained by first assigning the unequal variances flag to false. This can be done by setting the UnequalVariances property. The confidence interval can then be obtained by the LowerCIDiff and UpperCIDiff properties.

If the population variances are not equal, the ordinary \(t\) statistic does not have a \(t\) distribution and several approximate tests for the equality of means have been proposed. (See, for example, Anderson and Bancroft 1952, and Kendall and Stuart 1979.) One of the earliest tests devised for this situation is the Fisher-Behrens test, based on Fisher’s concept of fiducial probability. A procedure used in the TTest, LowerCIDiff and UpperCIDiff properties assuming unequal variances are specified is the Satterthwaite’s procedure, as suggested by H.F. Smith and modified by F.E. Satterthwaite (Anderson and Bancroft 1952, p. 83). Set UnequalVariances true to obtain results assuming unequal variances.

The test statistic is

\[
t' = (\bar{x}_1 - \bar{x}_2 - \mu_0) / s_d
\]

where

\[
s_d = \sqrt{\left(\frac{s_1^2}{n_1}\right) + \left(\frac{s_2^2}{n_2}\right)}
\]

Under the null hypothesis of \(\mu_1 - \mu_2 = c\), this quantity has an approximate \(t\) distribution with degrees of freedom \(df\), given by the following equation:
Inferences about Variances

The $F$ statistic for testing the equality of variances is given by $F = \frac{s^2_{\text{max}}}{s^2_{\text{min}}}$, where $s^2_{\text{max}}$ is the larger of $s^2_1$ and $s^2_2$. If the variances are equal, this quantity has an $F$ distribution with $n_1 - 1$ and $n_2 - 1$ degrees of freedom.

It is generally not recommended that the results of the $F$ test be used to decide whether to use the regular $t$-test or the modified $t'$ on a single set of data. The modified $t'$ (Satterthwaite’s procedure) is the more conservative approach to use if there is doubt about the equality of the variances.

Properties

ChiSquaredTest

```csharp
public double ChiSquaredTest {get; }
```

**Description**
The test statistic associated with the chi-squared test for common, or pooled, variances.

**Property Value**
A `double` containing the test statistic for the chi-squared test.

**Remarks**
The chi-squared test is a test of the hypothesis $\omega^2 = \omega^2_0$ where $\omega^2_0$ is the null hypothesis value as described in `ChiSquaredTestNull`.

ChiSquaredTestDF

```csharp
public int ChiSquaredTestDF {get; }
```

**Description**
The degrees of freedom associated with the chi-squared test for the common, or pooled, variances.

**Property Value**
A `int` which specifies the degrees of freedom for the chi-squared test.

**Remarks**
The chi-squared test is a test of the hypothesis $\omega^2 = \omega^2_0$ where $\omega^2_0$ is the null hypothesis value as described in `ChiSquaredTestNull`.

ChiSquaredTestNull

```csharp
public double ChiSquaredTestNull {get; set; }
```
Description
The null hypothesis value for the chi-squared test.

Property Value
A double containing the null hypothesis value for the chi-squared test.

Remarks
The default is 1.0.

ChiSquaredTestP
public double ChiSquaredTestP {get; }

Description
The probability of a larger chi-squared associated with the chi-squared test for common, or pooled, variances.

Property Value
A double containing the probability of a larger chi-squared for the chi-squared test for variances.

Remarks
The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where $\omega_0^2$ is the null hypothesis value as described in ChiSquaredTestNull.

ConfidenceMean
public double ConfidenceMean {get; set; }

Description
The confidence level (in percent) for a two-sided interval estimate of the mean of $x$ - the mean of $y$, in percent.

Property Value
A double containing the confidence level of the mean.

Remarks
ConfidenceMean must be between 0.0 and 1.0 and is often 0.90, 0.95 or 0.99. For a one-sided confidence interval with confidence level $c$ (at least 50 percent), set ConfidenceMean = $1.0 - 2.0(1.0 - c)$. If the confidence mean is not specified, a 95-percent confidence interval is computed, ConfidenceMean = .95.

ConfidenceVariance
public double ConfidenceVariance {get; set; }

Description
The confidence level (in percent) for two-sided interval estimate of the variances.

Property Value
A double containing the confidence level of the variance.
Remarks
Under the assumption of equal variances, the pooled variance is used to obtain a two-sided
ConfidenceVariance percent confidence interval for the common variance with
Imsl.Stat.NormTwoSample.LowerCICommonVariance (p. 570) or
Imsl.Stat.NormTwoSample.UpperCICommonVariance (p. 573). Without making the assumption of
equal variances, UnequalVariances (p. 572), the ratio of the variances is of interest. A two-sided
ConfidenceVariance percent confidence interval for the ratio of the variance of the first sample to that
of the second sample is given by the LowerCIRatioVariance and UpperCIRatioVariance. See
UnequalVariances (p. 572) and UpperCIRatioVariance (p. 573). The confidence intervals are
symmetric in probability. ConfidenceVariance must be between 0.0 and 1.0 and is often 0.90, 0.95 or
0.99. The default is 0.95.

DiffMean
public double DiffMean {get; }

Description
The difference of means for the two samples.

Property Value
A double containing the difference in mean.

Remarks
value = mean of \( x \) - mean of \( y \)

FTest
public double FTest {get; }

Description
The \( F \) test value of the \( F \) test for equality of variances.

Property Value
A double containing the \( F \) test value of the \( F \) test for equality of variances.

FTestDFdenominator
public int FTestDFdenominator {get; }

Description
The denominator degrees of freedom of the \( F \) test for equality of variances.

Property Value
A int containing the denominator degrees of freedom.

FTestDFnumerator
public int FTestDFnumerator {get; }

Description
The numerator degrees of freedom of the \( F \) test for equality of variances.
**Property Value**
A `int` containing the numerator degrees of freedom.

**FTestP**
```csharp
public double FTestP {get; }
```

**Description**
The probability of a larger $F$ in absolute value for the $F$ test for equality of variances, assuming equal variances.

**Property Value**
A `double` containing the probability of a larger $F$ in absolute value, assuming equal variances.

**LowerCICommonVariance**
```csharp
public double LowerCICommonVariance {get; }
```

**Description**
The lower confidence limits for the common, or pooled, variance.

**Property Value**
A `double` containing the lower confidence limits for the variance.

**LowerCIDiff**
```csharp
public double LowerCIDiff {get; }
```

**Description**
The lower confidence limit for the mean of the first population minus the mean of the second for equal or unequal variances.

**Property Value**
A `double` containing the lower confidence limit for the mean of the first sample minus the mean of the second sample.

**Remarks**
If UnequalVariances (p. 572) is `true` then the lower confidence limit for unequal variances will be returned.

**LowerCIRatioVariance**
```csharp
public double LowerCIRatioVariance {get; }
```

**Description**
The approximate lower confidence limit for the ratio of the variance of the first population to the second.

**Property Value**
A `double` containing the approximate lower confidence limit variance.

**MeanX**
```csharp
public double MeanX {get; }
```
Description
The mean of the first sample, $x$.

Property Value
A double containing the mean of $x$.

MeanY
public double MeanY {get; }

Description
The mean of the second sample, $y$.

Property Value
A double containing the mean of $y$.

PooledVariance
public double PooledVariance {get; }

Description
The Pooled variance for the two samples.

Property Value
A double containing the Pooled variance for the two samples.

StdDevX
public double StdDevX {get; }

Description
The standard deviation of the first sample, $x$.

Property Value
A double containing the standard deviation of the first sample, $x$.

StdDevY
public double StdDevY {get; }

Description
The standard deviation of the second sample, $y$.

Property Value
A double containing the standard deviation of the second sample, $y$.

TTest
public double TTest {get; }

Description
The test statistic for the Satterthwaite’s approximation for equal or unequal variances.

Property Value
A double containing the test statistic for the $t$-test.
Remarks
If UnequalVariances (p. 572) is true then the test statistic for unequal variances will be returned.

**TTestDF**
public double TTestDF {get; }

**Description**
The degrees of freedom for the Satterthwaite’s approximation for t-test for either equal or unequal variances.

**Property Value**
A double containing the degrees of freedom for the t-test.

**Remarks**
If UnequalVariances (p. 572) is true then the degrees of freedom for unequal variances will be returned.

**TTestNull**
public double TTestNull {get; set; }

**Description**
The Null hypothesis value for t-test for the mean.

**Property Value**
A double containing the hypothesis value.

**Remarks**
TTestNull = 0.0 by default.

**TTestP**
public double TTestP {get; }

**Description**
The approximate probability of a larger t for the Satterthwaite’s approximation for equal or unequal variances.

**Property Value**
A double containing the probability for the t-test.

**Remarks**
If UnequalVariances (p. 572) is true then the approximate probability of a larger t for unequal variances will be returned.

**UnequalVariances**
public bool UnequalVariances {get; set; }

**Description**
Specifies whether to return statistics based on equal or unequal variances.

**Property Value**
A bool which specifies whether the sample variances are unequal.
Remarks
A value of `true` will cause statistics for unequal variances to be returned. A value of `false` will cause statistics for equal variances to be returned. The default is to return statistics for equal variances.

**UpperCICommonVariance**
public double UpperCICommonVariance {get; }

Description
The upper confidence limits for the common, or pooled, variance.

Property Value
A `double` containing the upper confidence limits for the variance.

**UpperCIDiff**
public double UpperCIDiff {get; }

Description
The upper confidence limit for the mean of the first population minus the mean of the second for equal or unequal variances.

Property Value
A `double` containing the upper confidence limit for the mean of the first sample minus the mean of the second sample.

Remarks
If `UnequalVariances` (p. 572) is `true` then the upper confidence limit for unequal variances will be returned.

**UpperCIRatioVariance**
public double UpperCIRatioVariance {get; }

Description
The approximate upper confidence limit for the ratio of the variance of the first population to the second.

Property Value
A `double` containing the approximate upper confidence limit variance.

Constructor

**NormTwoSample**
public NormTwoSample(double[] x, double[] y)

Description
Constructor to compute statistics for mean and variance inferences using samples from two normal populations.
Parameters

- \( x \) – A double array containing the first sample.
- \( y \) – A double array containing the second sample.

Methods

\textbf{DowndateX}

\texttt{public void DowndateX(double[] \( x \))}

\textbf{Description}

Removes the observations in \( x \) from the first sample.

\textbf{Parameter}

- \( x \) – A double array containing the values to remove from the first sample.

\textbf{DowndateY}

\texttt{public void DowndateY(double[] \( y \))}

\textbf{Description}

Removes the observations in \( y \) from the second sample.

\textbf{Parameter}

- \( y \) – A double array containing the values to remove from the second sample.

\textbf{Update}

\texttt{public void Update(double[] \( x \), double[] \( y \))}

\textbf{Description}

Concatenates samples \( x \) and \( y \) to the samples provided in the constructor.

\textbf{Parameters}

- \( x \) – A double array containing updates to the first sample.
- \( y \) – A double array containing updates to the second sample.

\textbf{UpdateX}

\texttt{public void UpdateX(double[] \( x \))}

\textbf{Description}

Concatenates the values in \( x \) to the first sample provided in the constructor.

\textbf{Parameter}

- \( x \) – A double array containing updates for the first sample.

\textbf{UpdateY}

\texttt{public void UpdateY(double[] \( y \))}
Description
Concatenates the values in y to the second sample provided in the constructor.

Parameter
y – A double array containing updates for the second sample.

Example 1: NormTwoSample
This example taken from Conover and Iman(1983, p294), involves scores on arithmetic tests of two grade-school classes.

<table>
<thead>
<tr>
<th>Scores for Standard Group</th>
<th>Scores for Experimental Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>111</td>
</tr>
<tr>
<td>75</td>
<td>118</td>
</tr>
<tr>
<td>77</td>
<td>128</td>
</tr>
<tr>
<td>80</td>
<td>138</td>
</tr>
<tr>
<td>104</td>
<td>140</td>
</tr>
<tr>
<td>110</td>
<td>150</td>
</tr>
<tr>
<td>125</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>164</td>
</tr>
<tr>
<td></td>
<td>169</td>
</tr>
</tbody>
</table>

The question is whether a group taught by an experimental method has a higher mean score. The difference in means and the t test are output. The variances of the two populations are assumed to be equal. It is seen from the output that there is strong reason to believe that the two means are different (t value of -4.804). Since the lower 97.5-percent confidence limit does not include 0, the null hypothesis is that \( \mu_1 \leq \mu_2 \) would be rejected at the 0.05 significance level. (The closeness of the values of the sample variances provides some qualitative substantiation of the assumption of equal variances.)

```csharp
using System;
using Imsl.Stat;

public class NormTwoSampleEx1
{
    public static void Main(String[] args)
    {
        double mean;
        double[] x1 =
            new double[]{72.0, 75.0, 77.0, 80.0, 104.0, 110.0, 125.0};
        double[] x2 =
            new double[]{ 111.0, 118.0, 128.0, 138.0, 140.0,
                         150.0, 163.0, 164.0, 169.0};

        // Perform Analysis for one sample x2*/
        NormTwoSample n2samp = new NormTwoSample(x1, x2);
        mean = n2samp.DiffMean;

        Console.Out.WriteLine("x1mean-x2mean = " + mean);
        Console.Out.WriteLine("X1 mean =" + n2samp.MeanX);
        Console.Out.WriteLine("X2 mean =" + n2samp.MeanY);
    }
}
```
double pVar = n2samp.PooledVariance;
Console.Out.WriteLine("pooledVar = " + pVar);

double loCI = n2samp.LowerCIDiff;
double upCI = n2samp.UpperCIDiff;
Console.Out.WriteLine("95% CI for the mean is " + loCI + " " + upCI);

loCI = n2samp.LowerCIDiff;
upCI = n2samp.UpperCIDiff;
Console.Out.WriteLine("95% CI for the ueq mean is " + loCI + " " + upCI);

Console.Out.WriteLine("T Test Results");
double tDF = n2samp.TTestDF;
double tT = n2samp.TTest;
double tPval = n2samp.TTestP;
Console.Out.WriteLine("T default = " + tDF);
Console.Out.WriteLine("t = " + tT);
Console.Out.WriteLine("p-value = " + tPval);

double stdevX = n2samp.StdDevX;
double stdevY = n2samp.StdDevY;
Console.Out.WriteLine("stdev x1 =" + stdevX);
Console.Out.WriteLine("stdev x2 =" + stdevY);

Output

x1mean-x2mean = -50.4761904761905
X1 mean =91.8571428571428
X2 mean =142.333333333333
pooledVar = 434.632653061224
95% CI for the mean is -73.0100196252951 -27.9423613270859
95% CI for the ueq mean is -73.0100196252951 -27.9423613270859
T Test Results
T default = 14
t = -4.80436150471634
p-value = 0.000280258365677279
stdev x1 =20.8760514420118
stdev x2 =20.826655996585

Sort Class

public class Imsl.Stat.Sort

A collection of sorting functions.
Class Sort contains ascending and descending methods for sorting elements of an array or a matrix.

The QuickSort algorithm is used, except for short sequences which are handled using an insertion sort.

The QuickSort algorithm is a randomized QuickSort with 3-way partitioning. Basic QuickSort is slow if the sequence to be sorted contains many duplicate keys. The 3-way partitioning algorithm eliminates this problem. The pivot is chosen as the middle element of three potential pivots chosen at random.

The matrix ascending method sorts the rows of real matrix \( x \) using a particular row in \( x \) as the keys. The sort is algebraic with the first key as the most significant, the second key as the next most significant, etc.

When \( x \) is sorted in ascending order, the resulting sorted array is such that the following is true:

- For \( i = 0, 1, \ldots, n \) observations \( -2, x[i][\text{indices}_\text{keys}[0]] \leq x[i+1][\text{indices}_\text{keys}[0]] \)
- For \( k = 1, \ldots, \text{n\_keys} - 1, \) if \( x[i][\text{indices}_\text{keys}[j]] = x[i+1][\text{indices}_\text{keys}[j]] \) for \( j = 0, 1, \ldots, k - 1, \) then \( x[i][\text{indices}_\text{keys}[k]] = x[i+1][\text{indices}_\text{keys}[k]] \)

The observations also can be sorted in descending order. The rows of \( x \) containing the missing value code NaN in at least one of the specified columns are considered as an additional group. These rows are moved to the end of the sorted \( x \).

If all of the sort keys in a pair of rows are equal then the rows keep their original relative order.

The sorting algorithm is based on a quicksort method given by Singleton (1969) with modifications, see Bentley and Sedgewick (1997).

**Methods**

**Ascending**

static public void Ascending(int[] ra)

Description

Function to sort an integer array into ascending order.

Parameter

\( r a \) – int array to be sorted into ascending order

**Ascending**

static public void Ascending(int[] ra, int[] iperm)

Description

Sort an integer array into ascending order and returns the permutation vector.

Parameters

\( r a \) – int array to be sorted into ascending order

\( i p e r m \) – int is on input an array the same length as \( r a \). On output, it contains 0, 1, ..., sorted using the same permutations applied to \( r a \).
Ascending
static public void Ascending(double[] ra)

Description
Sort an array into ascending order.

Parameter
   ra – double array to be sorted into ascending order

Ascending
static public void Ascending(double[] ra, int[] iperm)

Description
Sort an array into ascending order and returns the permutation vector.

Parameters
   ra – double array to be sorted into ascending order
   iperm – int is on input an array the same length as ra. On output, it contains 0, 1, ..., sorted using the same permutations applied to ra.

Ascending
static public void Ascending(double[,] ra, int nKeys)

Description
Sort a matrix into ascending order by the first nKeys.

Parameters
   ra – double matrix to be sorted into ascending order.
   nKeys – int containing the first nKeys columns of ra to be used as the sorting keys.

Ascending
static public void Ascending(double[,] ra, int[] indkeys)

Description
Sort a matrix into ascending order by specified keys.

Parameters
   ra – a double matrix to be sorted into ascending order.
   indkeys – int array containing the order the columns of ra are to be sorted.

Ascending
static public void Ascending(double[,] ra, int nKeys, int[] iperm)

Description
Sort an array into ascending order by the first nKeys and returns the permutation vector.
Parameters
ra – double array to be sorted into ascending order.
nKeys – int containing the first nKeys columns of ra to be used as the sorting keys.
iperm – int is on input an array the same length as ra. On output, it contains 0, 1, ..., sorted using the same permutations applied to ra.

Ascending
static public void Ascending(double[,] ra, int[] indkeys, int[] iperm)

Description
Sort a matrix into ascending order by specified keys and returns the permutation vector.

Parameters
ra – double matrix to be sorted into ascending order.
indkeys – int array containing the order the columns of ra are to be sorted. These values must be between 0 and one less than the number of columns in ra.
iperm – int is on input an array the same length as ra. On output, it contains 0, 1, ..., sorted using the same permutations applied to ra.

Descending
static public void Descending(int[] ra)

Description
Function to sort an integer array into descending order.

Parameter
ra – int array to be sorted into descending order

Descending
static public void Descending(int[] ra, int[] iperm)

Description
Sort an integer array into descending order and returns the permutation vector.

Parameters
ra – int array to be sorted into descending order
iperm – int is on input an array the same length as ra. On output, it contains 0, 1, ..., sorted using the same permutations applied to ra.

Descending
static public void Descending(double[] ra)

Description
Sort an array into descending order.

Basic Statistics Sort • 579
**Parameter**

ra – double array to be sorted into descending order

---

**Descending**

static public void Descending(double[] ra, int[] iperm)

**Description**

Sort an array into descending order and returns the permutation vector.

**Parameters**

- ra – double array to be sorted into descending order
- iperm – int is on input an array the same length as ra. On output, it contains 0, 1, ..., sorted using the same permutations applied to ra.

---

**Descending**

static public void Descending(double[,] ra, int nKeys)

**Description**

Function to sort a matrix into descending order by the first nKeys.

**Parameters**

- ra – a double matrix to be sorted into descending order.
- nKeys – int containing the first nKeys columns of ra to be used as the sorting keys.

---

**Descending**

static public void Descending(double[,] ra, int[] indkeys)

**Description**

Function to sort a matrix into descending order by specified keys.

**Parameters**

- ra – a double matrix to be sorted into descending order.
- indkeys – int array containing the order the columns of ra are to be sorted.

---

**Descending**

static public void Descending(double[,] ra, int nKeys, int[] iperm)

**Description**

Function to sort an array into descending order by the first nKeys and returns the permutation vector.

**Parameters**

- ra – a double array to be sorted into descending order.
- nKeys – int containing the first nKeys columns of ra to be used as the sorting keys.
- iperm – an int array specifying the rearrangement (permutation) of the observations (rows) of ra.

---

**Descending**

static public void Descending(double[,] ra, int[] indkeys, int[] iperm)
Description
Function to sort a matrix into descending order by specified keys and return the permutation vector.

Parameters
ra – double matrix to be sorted into descending order.
indkeys – int array containing the order the columns of ra are to be sorted. These values must be between 0 and one less than the number of columns in ra.
iperm – an int array specifying the rearrangement (permutation) of the observations (rows) of ra.

Example 1: Sorting
An array is sorted by increasing value. A permutation array is also computed. Note that the permutation array begins at 0 in this example.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;

public class SortEx1
{
    public static void Main(String[] args)
    {
        double[] ra = new double[]{ 10.0, -9.0, 8.0, -7.0, 6.0,
                                    5.0, 4.0, -3.0, -2.0, -1.0};
        int[] iperm = new int[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

        PrintMatrix pm = new PrintMatrix("The Input Array");
        PrintMatrixFormat mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();
        // Print the array
        pm.Print(mf, ra);
        Console.Out.WriteLine();

        // Sort the array
        Sort.Ascending(ra, iperm);

        pm = new PrintMatrix("The Sorted Array - Lowest to Highest");
        mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();
        // Print the array
        pm.Print(mf, ra);

        pm = new PrintMatrix("The Resulting Permutation Array");
        mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();
        // Print the array
        pm.Print(mf, iperm);
    }
}
```
Output

The Input Array

10  
-9  
8  
-7  
6  
5  
4  
-3  
-2  
-1  

The Sorted Array - Lowest to Highest

-9  
-7  
-3  
-2  
-1  
4  
5  
6  
8  
10  

The Resulting Permutation Array

1  
3  
7  
8  
9  
6  
5  
4  
2  
0  

Example 2: Sorting

The rows of a 10 x 3 matrix x are sorted in ascending order using Columns 0 and 1 as the keys. There are two missing values (NaNs) in the keys. The observations containing these values are moved to the end of the sorted array.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;

class SortEx2
{

```
public static void Main(String[] args)
{
    int nKeys = 2;
    double[,] x = {
        {1.0, 1.0, 1.0}, {2.0, 1.0, 2.0},
        {1.0, 1.0, 3.0}, {1.0, 1.0, 4.0},
        {2.0, 2.0, 5.0}, {1.0, 2.0, 6.0},
        {1.0, 2.0, 7.0}, {1.0, 1.0, 8.0},
        {2.0, 2.0, 9.0}, {1.0, 1.0, 9.0}
    };

    int[] iperm = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    x[4,1] = Double.NaN;
    x[6,0] = Double.NaN;

    PrintMatrix pm = new PrintMatrix("The Input Array");
    PrintMatrixFormat mf = new PrintMatrixFormat();
    mf.SetNoRowLabels();
    mf.SetNoColumnLabels();
    // Print the array
    pm.Print(mf, x);
    Console.Out.WriteLine();

    Sort.Ascending(x, nKeys, iperm);

    pm = new PrintMatrix("The Sorted Array - Lowest to Highest");
    mf = new PrintMatrixFormat();
    mf.SetNoRowLabels();
    mf.SetNoColumnLabels();
    // Print the array
    pm.Print(mf, x);

    pm = new PrintMatrix("The permutation array");
    mf = new PrintMatrixFormat();
    mf.SetNoRowLabels();
    mf.SetNoColumnLabels();
    pm.Print(mf, iperm);
}

Output
The Input Array

1 1 1
2 1 2
1 1 3
1 1 4
2 NaN 5
1 2 6
NaN 2 7
1 1 8
2 2 9
1 1 9

Basic Statistics  Sort  583
The Sorted Array - Lowest to Highest

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>NaN</td>
<td>5</td>
</tr>
<tr>
<td>NaN</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

The permutation array

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

---

Ranks Class

```java
public class Imsl.Stat.Ranks
```

Compute the ranks, normal scores, or exponential scores for a vector of observations.

The class Ranks can be used to compute the ranks, normal scores, or exponential scores of the data in X. Ties in the data can be resolved in four different ways, as specified by property TieBreaker. The type of values returned can vary depending on the member function called:

**GetRanks: Ordinary Ranks**

For this member function, the values output are the ordinary ranks of the data in X. If X[i] has the smallest value among those in X and there is no other element in X with this value, then GetRanks(i) = 1. If both X[i] and X[j] have the same smallest value, then

- if TieBreaker = 0, Ranks[i] = GetRanks(j) = 1.5
- if TieBreaker = 1, Ranks[i] = Ranks[j] = 2.0
- if TieBreaker = 2, Ranks[i] = Ranks[j] = 1.0
if $\text{TieBreaker} = 3$, $\text{Ranks}[i] = 1.0$ and $\text{Ranks}[j] = 2.0$
or $\text{Ranks}[i] = 2.0$ and $\text{Ranks}[j] = 1.0$.

**GetBlomScores: Normal Scores, Blom Version**

Normal scores are expected values, or approximations to the expected values, of order statistics from a normal distribution. The simplest approximations are obtained by evaluating the inverse cumulative normal distribution function, $\text{CDF.InverseNormal}$, at the ranks scaled into the open interval $(0, 1)$. In the Blom version (see Blom 1958), the scaling transformation for the rank $r_i (1 \leq r_i \leq n)$, where $n$ is the sample size is $(r_i - 3/8)/(n + 1/4)$. The Blom normal score corresponding to the observation with rank $r_i$ is

$$
\Phi^{-1} \left( \frac{r_i - 3/8}{n + 1/4} \right)
$$

where $\Phi(\cdot)$ is the normal cumulative distribution function.

Adjustments for ties are made after the normal score transformation. That is, if $X[i]$ equals $X[j]$ (within fuzz) and their value is the $k$-th smallest in the data set, the Blom normal scores are determined for ranks of $k$ and $k + 1$, and then these normal scores are averaged or selected in the manner specified by $\text{TieBreaker}$, which is set by the property $\text{TieBreaker}$. (Whether the transformations are made first or ties are resolved first makes no difference except when averaging is done.)

**GetTukeyScores: Normal Scores, Tukey Version**

In the Tukey version (see Tukey 1962), the scaling transformation for the rank $r_i$ is $(r_i - 1/3)/(n + 1/3)$. The Tukey normal score corresponding to the observation with rank $r_i$ is

$$
\Phi^{-1} \left( \frac{r_i - 1/3}{n + 1/3} \right)
$$

Ties are handled in the same way as discussed above for the Blom normal scores.

**GetVanDerWaerdenScores: Normal Scores, Van der Waerden Version**

In the Van der Waerden version (see Lehmann 1975, page 97), the scaling transformation for the rank $r_i$ is $r_i/(n + 1)$. The Van der Waerden normal score corresponding to the observation with rank $r_i$ is

$$
\Phi^{-1} \left( \frac{r_i}{n + 1} \right)
$$

Ties are handled in the same way as discussed above for the Blom normal scores.

**GetNormalScores: Expected Value of Normal Order Statistics**

The member function $\text{GetNormalScores}$ returns the expected values of the normal order statistics. If the value in $X[i]$ is the $k$-th smallest, then the value output in $\text{SCORE}[i]$ is $E(Z_k)$, where $E(\cdot)$ is the expectation operator and $Z_k$ is the $k$-th order statistic in a sample of size $x.length$ from a standard normal distribution. Ties are handled in the same way as discussed above for the Blom normal scores.
GetSavageScores: Savage Scores

The member function GetSavageScores returns the expected values of the exponential order statistics. These values are called Savage scores because of their use in a test discussed by Savage (1956) (see Lehman 1975). If the value in $X[i]$ is the $k$-th smallest, then the $i$-th output value output is $E(Y_k)$, where $Y_k$ is the $k$-th order statistic in a sample of size $n$ from a standard exponential distribution. The expected value of the $k$-th order statistic from an exponential sample of size $n$ is

$$\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{n-k+1}$$

Ties are handled in the same way as discussed above for the Blom normal scores.

Properties

Fuzz

public double Fuzz {get; set; }

Description
The fuzz factor used in determining ties.

Property Value
A double which represents the fuzz factor.

Random

public System.Random Random {get; set; }

Description
The Random object.

Property Value
A Random object used in breaking ties.

TieBreaker

public Imsl.Stat.Ranks.Tie TieBreaker {get; set; }

Description
The tie breaker for Ranks.

Property Value
A Tie which represents the tie breaker.
**Constructor**

**Ranks**
public Ranks()

**Description**
Constructor for the Ranks class.

**Methods**

**ExpectedNormalOrderStatistic**
static public double ExpectedNormalOrderStatistic(int i, int n)

**Description**
Returns the expected value of a normal order statistic.

**Parameters**
- i – A int which specifies the rank of the order statistic.
- n – A int which specifies the sample size.

**Returns**
A double, the expected value of the i-th order statistic in a sample of size n from the standard normal distribution.

**GetBlomScores**
public double[] GetBlomScores(double[] x)

**Description**
Gets the Blom version of normal scores for each observation.

**Parameter**
- x – A double array which contains the observations to be ranked.

**Returns**
A double array which contains the Blom version of normal scores for each observation in x.

**GetNormalScores**
public double[] GetNormalScores(double[] x)

**Description**
Gets the expected value of normal order statistics (for tied observations, the average of the expected normal scores).

**Parameter**
- x – A double array which contains the observations.
Returns
A double array which contains the expected value of normal order statistics for the observations in x.

Remarks
For tied observations GetNormalScores returns an average of the expected normal scores.

GetRanks
public double[] GetRanks(double[] x)

Description
Gets the rank for each observation.

Parameter
x – A double array which contains the observations to be ranked.

Returns
A double array which contains the rank for each observation in x.

GetSavageScores
public double[] GetSavageScores(double[] x)

Description
Gets the Savage scores. (the expected value of exponential order statistics)

Parameter
x – A double array which contains the observations.

Returns
A double array which contains the Savage scores for the observations in x. (the expected value of exponential order statistics)

GetTukeyScores
public double[] GetTukeyScores(double[] x)

Description
Gets the Tukey version of normal scores for each observation.

Parameter
x – A double array which contains the observations to be ranked.

Returns
A double array which contains the Tukey version of normal scores for each observation in x.

GetVanDerWaerdenScores
public double[] GetVanDerWaerdenScores(double[] x)

Description
Gets the Van der Waerden version of normal scores for each observation.
Parameter

- A double array which contains the observations to be ranked.

Returns

A double array which contains the Van der Waerden version of normal scores for each observation in x.

Example: Ranks

In this data from Hinkley (1977) note that the fourth and sixth observations are tied and that the third and twentieth are tied.

using System;
using Imsl.Stat;
using Imsl.Math;

public class RanksEx1
{
    public static void Main(String[] args)
    {
        double[] x = new double[] {0.77, 1.74, 0.81, 1.20, 1.95, 1.20, 0.47, 1.43, 3.37, 2.20, 3.00, 3.09, 1.51, 2.10, 0.52, 1.62, 1.31, 0.32, 0.59, 0.81, 2.81, 1.87, 1.18, 1.35, 4.75, 2.48, 0.96, 1.89, 0.90, 2.05};

        PrintMatrixFormat mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();

        Ranks ranks = new Ranks();
        double[] score = ranks.GetRanks(x);
        new PrintMatrix("The Ranks of the Observations - " + "Ties Averaged").Print(mf, score);
        Console.Out.WriteLine();

        ranks = new Ranks();
        score = ranks.GetBlomScores(x);
        new PrintMatrix("The Blom Scores of the Observations - " + "Highest Score used in Ties").Print(mf, score);
        Console.Out.WriteLine();

        ranks = new Ranks();
        score = ranks.GetTukeyScores(x);
        new PrintMatrix("The Tukey Scores of the Observations - " + "Lowest Score used in Ties").Print(mf, score);
        Console.Out.WriteLine();

        ranks = new Ranks();
    }
}
random.Multipier = 16807;
ranks.Random = random;
score = ranks.GetVanDerWaerdenScores(x);
new PrintMatrix("The Van Der Waerden Scores of the " +
 "Observations - Ties untied by Random").Print(mf, score);
}

Output

The Ranks of the Observations - Ties Averaged

5
18
6.5
11.5
21
11.5
2
15
29
24
27
28
16
23
3
17
13
1
4
6.5
26
19
10
14
30
25
9
20
8
22

The Blom Scores of the Observations - Highest Score used in Ties

-1.02410618374162
0.208663745751154
-0.775546958322378
-0.294213138930921
0.472789120992267
-0.294213138930921
-1.60981606718445
-0.0414437330939986
1.60981606718445
0.7755469683322378
1.17581347255003
1.3608734286719
0.0414437332939865
0.6680021322269574
-1.3608734286719
0.124617407947998
-0.208663745751155
-2.04028133201041
-1.17581347255003
-0.7755469683322378
1.02410618374162
0.294213138930921
-0.472789120992267
-0.124617407947998
2.04028133201041
0.892918486444395
-0.56768639112746
0.381975767696542
-0.6680021322269574
0.56768639112746

The Tukey Scores of the Observations - Lowest Score used in Ties

-1.0200762327862
0.208082136154993
-0.88970115508476
-0.380874057516038
0.471389485888488
-0.380874057516038
-1.5986725959458
-0.0413298117473387
1.5986725959458
0.772935693128221
1.17060337087942
1.35372485367826
0.0413298117473387
0.665869518001049
-1.35372485367826
0.124273282084069
-0.208082136154993
-2.01450973381435
-1.17060337087942
0.88970115508476
1.0200762327862
0.2933812321121193
-0.471389485888488
-0.124273282084069
2.01450973381435
0.889701155084761
-0.565948821932863
0.380874057516038
-0.665869518001049
0.665948821932863

Basic Statistics

Ranks • 591
The Van Der Waerden Scores of the Observations - Ties untied by Random

-0.989168627340635
0.203544231532486
-0.864894358685283
-0.372289360465191
0.460494539103116
-0.286893916923039
-1.51792915959428
-0.0404405085656462
1.51792915959428
0.75272879425817
1.13097760824516
1.3001534336342
0.0404405085656462
0.649323913186466
-1.3001534336342
0.121587382750483
-0.203544231532486
-1.84859628850141
-1.13097760824516
-0.75272879425817
0.989168627340635
0.286893916923039
-0.460494539103116
-0.121587382750483
1.84859628850141
0.864894358685283
-0.552442584646774
0.372289360465191
-0.649323913186466
0.552442584646775

---

Ranks.Tie Enumeration

public enumeration Imsl.Stat.Ranks.Tie

Determines how to break a tie.

Fields

**Average**

public Imsl.Stat.Ranks.Tie Average
Description
Use the average score in the group of ties.

Highest
public Imsl.Stat.Ranks.Tie Highest

Description
Use the highest score in the group of ties.

Lowest
public Imsl.Stat.Ranks.Tie Lowest

Description
Use the lowest score in the group of ties.

Random
public Imsl.Stat.Ranks.Tie Random

Description
Use one of the group of ties chosen at random.

EmpiricalQuantiles Class

public class Imsl.Stat.EmpiricalQuantiles
Computes empirical quantiles.

The class EmpiricalQuantiles determines the empirical quantiles, as indicated in the array qProp, from the data in x. The algorithm first checks to see if x is sorted; if x is not sorted, the algorithm does either a complete or partial sort, depending on how many order statistics are required to compute the quantiles requested. The algorithm returns the empirical quantiles and, for each quantile, the two order statistics from the sample that are at least as large and at least as small as the quantile. For a sample of size n, the quantile corresponding to the proportion p is defined as

\[ Q(p) = (1 - f)x_{(j)} + fx_{(j+1)} \]

where \( j = \lfloor p(n+1) \rfloor \), \( f = p(n+1) - j \), and \( x_{(j)} \), is the j-th order statistic, if \( 1 \leq j \leq n \); otherwise, the empirical quantile is the smallest or largest order statistic.

Property

TotalMissing
public int TotalMissing {get; }

Basic Statistics EmpiricalQuantiles • 593
Description
The total number of missing values.

Property Value
An int representing the total number of missing values (NaN) in input x.

Constructor

EmpiricalQuantiles
public EmpiricalQuantiles(double[] x, double[] qProp)

Description
Computes empirical quantiles.

Parameters
x – A double array containing the data.
qProp – A double array containing the quantile proportions.

Methods

GetQ
public double[] GetQ()

Description
Returns the empirical quantiles.

Returns
A double array of length qProp.Length containing the empirical quantiles.

Remarks
Q[i] corresponds to the empirical quantile at proportion qProp[i]. The quantiles are determined by linear interpolation between adjacent ordered sample values.

GetXHi
public double[] GetXHi()

Description
Returns the smallest element of x greater than or equal to the desired quantile.
Returns
A double array of length qProp.Length containing the smallest element of x greater than or equal to the desired quantile.

GetXLo
public double[] GetXLo()

Description
Returns the largest element of x less than or equal to the desired quantile.

Returns
A double array of length qProp.Length containing the largest element of x less than or equal to the desired quantile.

Example: Empirical Quantiles
In this example, five empirical quantiles from a sample of size 30 are obtained. Notice that the 0.5 quantile corresponds to the sample median. The data are from Hinkley (1977) and Velleman and Hoaglin (1981). They are the measurements (in inches) of precipitation in Minneapolis/St. Paul during the month of March for 30 consecutive years.

```csharp
using System;
using Imsl.Stat;

class EmpiricalQuantilesEx1
{
    public static void Main(string[] args)
    {
        double[] x = {0.77, 1.74, 0.81, 1.20, 1.95, 1.20, 0.47, 1.43, 3.37, 2.20, 3.00, 3.09, 1.51, 2.10, 0.52, 1.62, 1.31, 0.32, 0.59, 0.81, 2.81, 1.87, 1.18, 1.35, 4.75, 2.48, 0.96, 1.89, 0.90, 2.05};
        double[] qProp = {0.01, 0.5, 0.90, 0.95, 0.99};
        EmpiricalQuantiles eq = new EmpiricalQuantiles(x, qProp);

        double[] Q = eq.GetQ();
        double[] XLo = eq.GetXLo();
        double[] XHi = eq.GetXHi();

        Console.WriteLine(" Smaller Empirical Larger");
        Console.WriteLine(" Quantile Datum Quantile Datum");
        for (int i = 0; i < qProp.Length; i++)
            Console.WriteLine(" {0}\t\{1}\t\{2}\t\{3}\t", qProp[i], XLo[i], Q[i], XHi[i]);
    }
}
```

Output

<table>
<thead>
<tr>
<th>Quantile</th>
<th>Smaller Datum</th>
<th>Empirical Quantile</th>
<th>Larger Datum</th>
</tr>
</thead>
</table>

Basic Statistics

EmpiricalQuantiles • 595
TableOneWay Class

public class Imsl.Stat.TableOneWay

Tallies observations into a one-way frequency table.

Class TableOneWay calculates a frequency table for a data array.

A one-way frequency table can be used to visualize the shape of the data distribution and look for anomalies in the data. There are many approaches to constructing frequency tables. Four approaches are implemented in this class:

1. equal width class intervals based upon the smallest and largest observations,
2. equal width class intervals based upon a user provided minimum and maximum,
3. class intervals defined from user provided class midpoints, and
4. class intervals defined from user provided class boundaries.

The TableOneWay class implements the first two approaches by overloading the GetFrequencyTable method. If GetFrequencyTable() is used without input arguments, nIntervals of equal length are formed between the minimum and maximum values in the data. The frequency table returned from this method contains tallies of the number of observations in each interval. The data minimum and maximum can be obtained from the Minimum and Maximum properties.

Instead of using the minimum and maximum to define the boundaries of the smallest and largest classes, specified boundaries can be used by calling GetFrequencyTable(lower_bound, upper_bound). This method tallies all data less than or equal to the lower_bound into the first class, and all data greater than or equal to upper_bound into the last class.

The third approach is implemented using the GetFrequencyTableUsingClassmarks method. Equally spaced intervals can be defined using class marks. In this approach a double precision array of length nIntervals containing the class midpoints is passed to the GetFrequencyTableUsingClassmarks(classmarks[]). The class marks, or midpoints, must be equally spaced.

Finally in those applications where unequal length intervals are preferred, the GetFrequencyTableUsingCutpoints(cutpoints[]) method can be used. The double precision array cutpoints has length nIntervals-1 and contains the class boundaries listed in ascending order. The first cut point defines the first class which is used to tally all data less than or equal to the first cut
point value. The last cut point defines the last class which is used to tally all data greater than or equal to the last cut point value.

Properties

Maximum
public double Maximum {get; }

Description
The maximum value of x.

Property Value
A double containing the maximum data bound.

Minimum
public double Minimum {get; }

Description
The minimum value of x.

Property Value
A double containing the minimum data bound.

Constructor

TableOneWay
public TableOneWay(double[] x, int nIntervals)

Description
Constructor for TableOneWay.

Parameters
x – A double array containing the observations.
nIntervals – A int scalar containing the number of intervals (bins).

Methods

GetFrequencyTable
public double[] GetFrequencyTable()
Description
Returns the one-way frequency table.

Returns
A double array containing the one-way frequency table.

Remarks
\( n \text{Intervals} \) intervals of equal length are used with the initial interval starting with the minimum value in \( x \) and the last interval ending with the maximum value in \( x \). The initial interval is closed on the left and the right. The remaining intervals are open on the left and closed on the right. Each interval is of length \( (\text{max} - \text{min})/n \text{Intervals} \), where \( \text{max} \) is the maximum value of \( x \) and \( \text{min} \) is the minimum value of \( x \).

GetFrequencyTable
public double[] GetFrequencyTable(double lowerBound, double upperBound)

Description
Returns a one-way frequency table using known bounds.

Parameters
- lowerBound – A double specifies the right endpoint.
- upperBound – A double specifies the left endpoint.

Returns
A double array containing the one-way frequency table.

Remarks
The one-way frequency table is computed using two semi-infinite intervals as the initial and last intervals. The initial interval is closed on the right and includes lowerBound as its right endpoint. The last interval is open on the left and includes all values greater than upperBound. The remaining \( n \text{Intervals} - 2 \) intervals are each of length \( (\text{upperBound} - \text{lowerBound}) / (n \text{Intervals} - 2) \) and are open on the left and closed on the right. \( n \text{Intervals} \) must be greater than or equal to 3.

GetFrequencyTableUsingClassmarks
public double[] GetFrequencyTableUsingClassmarks(double[] classmarks)

Description
Returns the one-way frequency table using class marks.

Parameter
- classmarks – A double array containing either the cutpoints or the class marks.

Returns
A double array containing the one-way frequency table.
Remarks
Equally spaced class marks in ascending order must be provided in the array classmarks of length nIntervals. The class marks are the midpoints of each of the nIntervals. Each interval is assumed to have length classmarks[1] - classmarks[0]. nIntervals must be greater than or equal to 2.

GetFrequencyTableUsingCutpoints
public double[] GetFrequencyTableUsingCutpoints(double[] cutpoints)

Description
Returns the one-way frequency table using cutpoints.

Parameter
- cutpoints – A double array containing the cutpoints.

Returns
A double array containing the one-way frequency table.

Remarks
The cutpoints are boundaries that must be provided in the array cutpoints of length nIntervals-1. This option allows unequal interval lengths. The initial interval is closed on the right and includes the initial cutpoint as its right endpoint. The last interval is open on the left and includes all values greater than the last cutpoint. The remaining nIntervals-2 intervals are open on the left and closed on the right. Argument nIntervals must be greater than or equal to 3 for this option.

Example: TableOneWay
The data for this example is from Hinkley (1977) and Belleman and Hoaglin (1981). The measurements (in inches) are for precipitation in Minneapolis/St. Paul during the month of March for 30 consecutive years.

The first test uses the default tally method which may be appropriate when the range of data is unknown. The minimum and maximum data bounds are displayed.

The second test computes the table using known bounds, where the lower bound is 0.5 and the upper bound is 4.5. The eight interior intervals each have width (4.5 - 0.5)/(10-2) = 0.5. The 10 intervals are \((-\infty, 0.5], (0.5,1.0],..., (4.0,4.5], \) and \((4.5,\infty]\).

In the third test, 10 class marks, 0.25, 0.75, 1.25,...,4.75, are input. This defines the class intervals \((0.0,0.5],(0.5,1.0],..., (4.0,4.5],[4.5,5.0]\). Note that unlike the previous test, the initial and last intervals are the same length as the remaining intervals.

In the fourth test, cutpoints, 0.5,1.0, 1.5, 2.0, ...,4.5, are input to define the same 10 intervals as in the second test. Here again, the initial and last intervals are semi-infinite intervals.

class TableOneWayEx1
{
    public static void Main(String[] args)
    {
        // Code here
    }
}
```csharp
int nIntervals = 10;

double[] x = new double[]{ 0.77, 1.74, 0.81, 1.20, 1.95,
                          1.20, 0.47, 1.43, 3.37, 2.20,
                          3.00, 3.09, 0.59, 0.81,
                          2.81, 1.87, 1.18, 1.35, 4.75,
                          2.48, 0.96, 1.89, 0.9, 2.05};
double[] cutPoints = new double[]{ 0.5, 1.0, 1.5, 2.0, 2.5,
                                    3.0, 3.5, 4.0, 4.5};
double[] classMarks = new double[]{ 0.25, 0.75, 1.25, 1.75,
                                    2.25, 2.75, 3.25, 3.75,
                                    4.25, 4.75};

TableOneWay fTbl = new TableOneWay(x, nIntervals);

double[] table = fTbl.GetFrequencyTable();

Console.Out.WriteLine("Example 1");
for (int i = 0; i < table.Length; i++)
    Console.Out.WriteLine(i + "
                          " + table[i]);

Console.Out.WriteLine("---------------");
Console.Out.WriteLine("Lower bounds= " + fTbl.Minimum);
Console.Out.WriteLine("Upper bounds= " + fTbl.Maximum);

// getFrequencyTable using a set of known bounds */
// table = fTbl.GetFrequencyTable(0.5, 4.5);
for (int i = 0; i < table.Length; i++)
    Console.Out.WriteLine(i + "
                          " + table[i]);

Console.Out.WriteLine("---------------");

table = fTbl.GetFrequencyTableUsingClassmarks(classMarks);
for (int i = 0; i < table.Length; i++)
    Console.Out.WriteLine(i + "
                          " + table[i]);

Console.Out.WriteLine("---------------");

table = fTbl.GetFrequencyTableUsingCutpoints(cutPoints);
for (int i = 0; i < table.Length; i++)
    Console.Out.WriteLine(i + "
                          " + table[i]);
```

**Output**

Example 1

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
TableTwoWay Class

public class Imsl.Stat.TableTwoWay

Tallies observations into a two-way frequency table.

Class TableTwoWay calculates a two-dimensional frequency table for a data array based upon two variables.

A two-way frequency table can be used to visualize the shape of the bivariate distribution and look for
anomalies in the data. There are many approaches to constructing two-way frequency tables. Four approaches are implemented in this class:

1. equal width class intervals based upon the smallest and largest observations,
2. equal width class intervals based upon a user provided minimum and maximum,
3. class intervals defined from user provided class midpoints, and
4. class intervals defined from user provided class boundaries.

The TableTwoWay class implements the first two approaches by overloading the GetFrequencyTable method. If GetFrequencyTable() is used without input arguments, `xIntervals` intervals of equal length are formed between the minimum and maximum values in `x`, and similarly, `yIntervals` intervals are formed for `y`. The frequency table returned from this method contains tallies of the number of observations in each interval. The data minimum and maximum can be obtained using the `MinimumX`, `MinimumY`, `MaximumX` and `MaximumY` properties.

Instead of using the minimum and maximum to define the boundaries of the smallest and largest classes, specified boundaries can be used by calling GetFrequencyTable(xLowerBound, xUpperBound, yLowerBound, yUpperBound). This method tallies all data less than or equal to the `xLowerBound` and `yLowerBound` into the first class, and all data greater than or equal to `xUpperBound` and `yUpperBound` into the last class.

The third approach is implemented using the GetFrequencyTableUsingClassmarks method. Equally spaced intervals can be defined using class marks. In this approach two double precision arrays of length `xIntervals` and `yIntervals` containing the class midpoints for `x` and `y` respectively are passed to the GetFrequencyTableUsingClassmarks(cx[], cy[]). The class marks, or midpoints, must be equally spaced.

Finally in those applications where unequal length intervals are preferred, the GetFrequencyTableUsingCutpoints(cx[], cy[]) method can be used. The double precision arrays `cx` and `cy` with lengths `xIntervals-1` and `yIntervals-1` respectively contain the class boundaries listed in ascending order. The first cut point defines the first class which is used to tally all data less than or equal to the first cut point value. The last cut point defines the last class which is used to tally all data greater than or equal to the last cut point value.

**Properties**

**MaximumX**

`public double MaximumX {get; }`

**Description**

The maximum value of `x.`

---

602 • TableTwoWay  IMSL C# Numerical Library
**Property Value**

A double containing the maximum data bound for x.

**MaximumY**

public double MaximumY {get; }

**Description**

The maximum value of y.

**Property Value**

A double containing the maximum data bound for y.

**MinimumX**

public double MinimumX {get; }

**Description**

The minimum value of x.

**Property Value**

A double containing the minimum data bound for x.

**MinimumY**

public double MinimumY {get; }

**Description**

The minimum value of y.

**Property Value**

A double containing the minimum data bound for y.

---

**Constructor**

**TableTwoWay**

public TableTwoWay(double[] x, int xIntervals, double[] y, int yIntervals)

**Description**

Constructor for TableTwoWay.

**Parameters**

- x – A double array containing the data for the first variable.
- xIntervals – A int scalar containing the number of intervals (bins) for variable x.
- y – A double array containing the data for the second variable.
- yIntervals – A int scalar containing the number of intervals (bins) for variable y.
Methods

GetFrequencyTable
public double[,] GetFrequencyTable()

Description
Returns the two-way frequency table.

Returns
A two-dimensional double array containing the two-way frequency table.

Remarks
Intervals of equal length are used. Let $x_{min}$ and $x_{max}$ be the minimum and maximum values in x, respectively, with similar meanings for $y_{min}$ and $y_{max}$. Then, the first row of the output table is the tally of observations with the x value less than or equal to $x_{min} + \frac{(x_{max} - x_{min})}{x_{Intervals}}$, and the y value less than or equal to $y_{min} + \frac{(y_{max} - y_{min})}{y_{Intervals}}$.

GetFrequencyTable
public double[,] GetFrequencyTable(double xLowerBound, double xUpperBound, double yLowerBound, double yUpperBound)

Description
Compute a two-way frequency table using intervals of equal length and user supplied upper and lower bounds, $x_{LowerBound}$, $x_{UpperBound}$, $y_{LowerBound}$, $y_{UpperBound}$.

Parameters
- $x_{LowerBound}$ – A double specifies the right endpoint for x.
- $x_{UpperBound}$ – A double specifies the left endpoint for x.
- $y_{LowerBound}$ – A double specifies the right endpoint for y.
- $y_{UpperBound}$ – A double specifies the left endpoint for y.

Returns
A two dimensional double array containing the two-way frequency table.

Remarks
The first and last intervals for both variables are semi-infinite in length. $x_{Intervals}$ and $y_{Intervals}$ must be greater than or equal to 3.

GetFrequencyTableUsingClassmarks
public double[,] GetFrequencyTableUsingClassmarks(double[] cx, double[] cy)

Description
Returns the two-way frequency table using class marks.

Parameters
- cx – A double array containing the class marks for x.
- cy – A double array containing the class marks for y.
Returns
A two dimensional double array containing the two-way frequency table.

Remarks
Class marks are the midpoints of $xIntervals$ and $yIntervals$. Equally spaced class marks in ascending order must be provided in the arrays $cx$ and $cy$. The class marks the midpoints of each interval. Each interval is taken to have length $cx[1] - cx[0]$ in the $x$ direction and $cy[1] - cy[0]$ in the $y$ direction. The total number of elements in the output table may be less than the number of observations of input data. Arguments $xIntervals$ and $yIntervals$ must be greater than or equal to 2 for this option.

**GetFrequencyTableUsingCutpoints**

```java
public double[,] GetFrequencyTableUsingCutpoints(double[] cx, double[] cy)
```

Description
Returns the two-way frequency table using cutpoints.

Parameters
- $cx$ – A double array containing the cutpoints for $x$.
- $cy$ – A double array containing the cutpoints for $y$.

Returns
A two dimensional double array containing the two-way frequency table.

Remarks
The cutpoints (boundaries) must be provided in the arrays $cx$ and $cy$, of length ($xIntervals-1$) and ($yIntervals-1$) respectively. The first row of the output table is the tally of observations for which the $x$ value is less than or equal to $cx[0]$, and the $y$ value is less than or equal to $cy[0]$. This option allows unequal interval lengths. Arguments $cx$ and $cy$ must be greater than or equal to 2.

Example: TableTwoWay
The data for $x$ in this example is from Hinkley (1977) and Belleman and Hoaglin (1981). The measurements (in inches) are for precipitation in Minneapolis/St. Paul during the month of March for 30 consecutive years. The data for $y$ were created by adding small integers to the data in $x$.

The first test uses the default tally method which may be appropriate when the range of data is unknown. The minimum and maximum data bounds are displayed.

The second test computes the table using known bounds, where the $x$ lower, $x$ upper, $y$ lower, $y$ upper bounds are chosen so that the intervals will be 0 to 1, 1 to 2, and so on for $x$ and 1 to 2, 2 to 3 and so on for $y$.

In the third test, the class boundaries are input at the same intervals as in the second test. The first element of $cmx$ and $cmy$ specify the first cutpoint between classes.

The fourth test uses the cutpoints tally option with cutpoints such that the intervals are specified as in the previous tests.
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class TableTwoWayEx1
{
    public static void Main(String[] args)
    {
        int nx = 5;
        int ny = 6;
        double[] x = new double[]{ 0.77, 1.74, 0.81, 1.20, 1.95,
            1.20, 0.47, 1.43, 3.37, 2.20,
            3.00, 3.09, 1.51, 2.10, 0.52,
            1.62, 1.31, 0.32, 0.59, 0.81,
            2.81, 1.87, 1.18, 1.35, 4.75,
            2.48, 0.96, 1.89, 0.9, 2.05};
        double[] y = new double[]{ 1.77, 3.74, 3.81, 2.20, 3.95,
            4.20, 1.47, 3.43, 6.37, 3.20,
            5.00, 6.09, 2.51, 4.10, 3.52,
            2.62, 3.31, 3.32, 1.59, 2.81,
            5.81, 2.87, 3.18, 4.35, 5.75,
            4.48, 3.96, 2.89, 2.9, 5.05};
        TableTwoWay fTbl = new TableTwoWay(x, nx, y, ny);
        double[,] table = fTbl.GetFrequencyTable();
        Console.Out.WriteLine("Example 1");
        Console.Out.WriteLine("Use Min and Max for bounds");
        new PrintMatrix("counts").Print(table);
        Console.Out.WriteLine("--------------------------");
        Console.Out.WriteLine("Lower xbounds= " + fTbl.MinimumX);
        Console.Out.WriteLine("Upper xbounds= " + fTbl.MaximumX);
        Console.Out.WriteLine("Lower ybounds= " + fTbl.MinimumY);
        Console.Out.WriteLine("Upper ybounds= " + fTbl.MaximumY);
        Console.Out.WriteLine("--------------------------");
        double xlo = 1.0;
        double xhi = 4.0;
        double ylo = 2.0;
        double yhi = 6.0;
        Console.Out.WriteLine("Use Known bounds");
        table = fTbl.GetFrequencyTable(xlo, xhi, ylo, yhi);
        new PrintMatrix("counts").Print(table);
        double[] cmx = new double[]{0.5, 1.5, 2.5, 3.5, 4.5};
        double[] cmy = new double[]{1.5, 2.5, 3.5, 4.5, 5.5, 6.5};
        table = fTbl.GetFrequencyTableUsingClassmarks(cmx, cmy);
        Console.Out.WriteLine("Use Class Marks");
        new PrintMatrix("counts").Print(table);
        double[] cpix = new double[]{1, 2, 3, 4};
double[] cpy = new double[]{2, 3, 4, 5, 6};
table = fTbl.GetFrequencyTableUsingCutpoints(cpx, cpy);
Console.Out.WriteLine("Use Cutpoints");
new PrintMatrix("counts").Print(table);

Output

Example 1
Use Min and Max for bounds

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

-------------------------------
Lower xbounds= 0.32
Upper xbounds= 4.75
Lower ybounds= 1.47
Upper ybounds= 6.37
-------------------------------

Use Known bounds

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Use Class Marks

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Use Cutpoints

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
TableMultiWay Class

public class Imsl.Stat.TableMultiWay

Tallies observations into a multi-way frequency table.

The TableMultiWay class determines the distinct values in multivariate data and computes frequencies for the data. This class accepts the data in the matrix x, but performs computations only for the variables (columns) in the first nkeys columns of x or by the variables specified in indkeys. In general, the variables for which frequencies should be computed are discrete; they should take on a relatively small number of different values. Variables that are continuous can be grouped first. TableMultiWay can be used to group variables and determine the frequencies of groups.

The read-only property BalancedTable returns a TableBalanced object. Its GetValues method returns an array with the unique values in the vector of the variables and tallies the number of unique values of each variable table. Each combination of one value from each variable forms a cell in a multi-way table. The frequencies of these cells are entered in a table so that the first variable cycles through its values exactly once, and the last variable cycles through its values most rapidly. Some cells cannot correspond to any observations in the data; in other words, “missing cells” are included in table and have a value of 0.

The read-only property UnbalancedTable returns a TableUnbalanced object. The frequency of each cell is entered in the unbalanced table so that the first variable cycles through its values exactly once and the last variable cycles through its values most rapidly. table is returned by UnbalancedTable property. All cells have a frequency of at least 1, i.e., there is no “missing cell.” The array listCells, returned by method GetListCells can be considered “parallel” to table because row i of listCells is the set of nkeys values that describes the cell for which row i of table contains the corresponding frequency.

Properties

BalancedTable
public Imsl.Stat.TableMultiWay.TableBalanced BalancedTable {get; }

Description
An object containing the balanced table.

Property Value
A TableMultiWay.TableBalanced object.
UnbalancedTable
public Imsl.Stat.TableMultiWay.TableUnbalanced UnbalancedTable {get; }

Description
An object containing the unbalanced table.

Property Value
A TableMultiWay.TableUnBalanced object.

Constructors

TableMultiWay
public TableMultiWay(double[,] x, int nKeys)

Description
Constructor for TableMultiWay.

Parameters
  x – A double matrix containing the observations and variables.
  nKeys – A int array containing the variables(columns) for which computations are to be performed.

TableMultiWay
public TableMultiWay(double[,] x, int[] indkeys)

Description
Constructor for TableMultiWay.

Parameters
  x – A double matrix containing the observations and variables.
  indkeys – A int array containing the variables(columns) for which computations are to be performed.

Methods

GetGroups
public int[] GetGroups()

Description
Returns the number of observations (rows) in each group.

Returns
A int array containing the number of observations (row) in each group.
Remarks
The number of groups is the length of the returned array. A group contains observations in x that are
equal with respect to the method of comparison. If n contains the returned integer array, then the first
n[0] rows of the sorted x are group number 1. The next n[1] rows of the sorted x are group number 2,
etc. The last n[n.length - 1] rows of the sorted x are group number n.length.

SetFrequencies
public void SetFrequencies(double[] frequencies)

Description
Sets the frequencies for each observation in x.

Parameter
frequencies – A double array containing the frequency for each observation in x.

Remarks
Length of input must be the same as the number of observations or number of rows in x.
By default, frequencies[] = 1.

Example 1: TableMultiWay
The same data used in SortEx2 is used in this example. It is a 10 x 3 matrix using Columns 0 and 1 as
keys. There are two missing values (NaNs) in the keys. NaN is displayed as a ?. Table MultiWay
determines the number of groups of different observations.

using System;
using Imsl.Stat;
using Imsl.Math;

public class TableMultiWayEx1
{
    public static void Main(String[] args)
    {
        int nKeys = 2;
        double[,] x = {
            {1.0, 1.0, 1.0}, {2.0, 1.0, 2.0},
            {1.0, 1.0, 3.0}, {1.0, 1.0, 4.0},
            {2.0, 2.0, 5.0}, {1.0, 2.0, 6.0},
            {1.0, 2.0, 7.0}, {1.0, 1.0, 8.0},
            {2.0, 2.0, 9.0}, {1.0, 1.0, 9.0}};

        x[4,1] = Double.NaN;
        x[6,0] = Double.NaN;

        PrintMatrix pm = new PrintMatrix("The Input Array");
        PrintMatrixFormat mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();
        // Print the array
        pm.Print(mf, x);
        Console.Out.WriteLine();
TableMultiWay tbl = new TableMultiWay(x, nKeys);
int[] ngroups = tbl.GetGroups();
Console.Out.WriteLine(" ngroups");
for (int i = 0; i < ngroups.Length; i++)
    Console.Out.Write(ngroups[i] + " ");

Output
The Input Array
1 1 1
2 1 2
1 1 3
1 1 4
2 NaN 5
1 2 6
NaN 2 7
1 1 8
2 2 9
1 1 9

ngroups
5 1 1 1

Example 2: TableMultiWay
The table of frequencies for a data matrix of size 30 x 2 is output.

using System;
using Imsl.Stat;
using Imsl.Math;

public class TableMultiWayEx2
{
    public static void Main(String[] args)
    {
        int[] indkeys = new int[]{0, 1};
        double[,] x = {
            {0.5, 1.5}, {1.5, 3.5},
            {0.5, 3.5}, {1.5, 2.5},
            {1.5, 3.5}, {1.5, 4.5},
            {0.5, 1.5}, {1.5, 3.5},
            {3.5, 6.5}, {2.5, 3.5},
            {2.5, 4.5}, {3.5, 6.5},
            {1.5, 2.5}, {2.5, 4.5},
            {0.5, 3.5}, {1.5, 2.5},
            {1.5, 3.5}, {0.5, 3.5},
            {0.5, 1.5}, {0.5, 2.5},
            {2.5, 5.5}, {1.5, 2.5},
            {1.5, 3.5}, {1.5, 4.5},
        };
    }
}
TableMultiWay tbl = new TableMultiWay(x, indkeys);

int[] nvalues = tbl.BalancedTable.GetNvalues();

double[] values = tbl.BalancedTable.GetValues();

Console.Out.WriteLine(" row values");
for (int i = 0; i < nvalues[0]; i++)
    Console.Out.Write(values[i] + " ");
Console.Out.WriteLine(" column values");
for (int i = 0; i < nvalues[1]; i++)
    Console.Out.Write(values[i + nvalues[0]] + " ");

double[] table = tbl.BalancedTable.GetTable();

Console.Out.WriteLine(" Table");

Console.Out.WriteLine(" ");
for (int i = 0; i < nvalues[1]; i++)
    Console.Out.Write(values[i + nvalues[0]] + " ");
Console.Out.WriteLine(" ");
for (int i = 0; i < nvalues[0]; i++)
{
    Console.Out.Write(values[i] + " ");
    for (int j = 0; j < nvalues[1]; j++)
        Console.Out.Write(table[j + (nvalues[1] * i)] + " ");
    Console.Out.WriteLine(" ");
}

Output

row values
0.5 1.5 2.5 3.5 4.5

column values
1.5 2.5 3.5 4.5 5.5 6.5

Table

1.5 2.5 3.5 4.5 5.5 6.5
0.5 3 2 4 0 0 0
1.5 0 5 5 2 0 0
2.5 0 0 1 3 2 0
Example 3: TableMultiWay

The unbalanced table of frequencies for a data matrix of size 4 x 3 is output.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;

public class TableMultiWayEx3
{
    public static void Main(String[] args)
    {
        int[] indkeys = new int[]{0, 1};
        double[,] x = {
            {2.0, 5.0, 1.0}, {1.0, 5.0, 2.0},
            {1.0, 6.0, 3.0}, {2.0, 6.0, 4.0}};
        double[] frq = new double[]{1.0, 2.0, 3.0, 4.0};

        TableMultiWay tbl = new TableMultiWay(x, indkeys);
        tbl.SetFrequencies(frq);

        int ncells = tbl.UnbalancedTable.NCells;
        double[] listCells = tbl.UnbalancedTable.GetListCells();
        double[] table = tbl.UnbalancedTable.GetTable();

        PrintMatrix pm = new PrintMatrix("List Cells");
        PrintMatrixFormat mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        // Print the array
        pm.Print(mf, listCells);
        Console.Out.WriteLine();

        pm = new PrintMatrix("Unbalanced Table");
        mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();
        // Print the array
        pm.Print(mf, table);
        Console.Out.WriteLine();
    }
}
```

Output

List Cells

1
5
1
6
TableMultiWay.TableBalanced Class

public class Imsl.Stat.TableMultiWay.TableBalanced

Tallies the number of unique values of each variable.

Methods

GetNvalues

public int[] GetNvalues()

Description

Returns an array of length nkeys containing in its i-th element (i=0,1,...,nkeys-1), the number of levels or categories of the i-th classification variable (column).

Returns

A int array containing the number of levels or for each variable (column) in x.

GetTable

public double[] GetTable()

Description

Returns an array containing the frequencies for each variable.

Returns

A double array containing the frequencies for each variable in x.
Remarks
The array is of length nValues[0] x nValues[1] x ... x nValues[nkeys] containing the frequencies in the cells of the table to be fit, where nValues contains the result from getNValues.

Empty cells are included in table, and each element of table is nonnegative. The cells of table are sequenced so that the first variable cycles through its nValues[0] categories one time, the second variable cycles through its nValues[1] categories nValues[0] times, the third variable cycles through its nValues[2] categories nValues[0] * nValues[1] times, etc., up to the nkeys-th variable, which cycles through its nValues[nkeys - 1] categories nValues[0] * nValues[1] * ... * nValues[nkeys - 2] times.

GetValues
public double[] GetValues()

Description
Returns the values of the classification variables.

Returns
A double array containing the values of the classification variables.

Remarks
GetValues returns an array of length nValues[0] + nValues[1] + ... + nValues[nkeys - 1]. The first nValues[0] elements contain the values for the first classification variable. The next nValues[1] contain the values for the second variable. The last nValues[nkeys - 1] positions contain the values for the last classification variable, where nValues contains the result from getNValues.

TableMultiWay.TableUnbalanced Class

public class Imsl.Stat.TableMultiWay.TableUnbalanced

Tallies the frequency of each cell in x.

Property

NCells
public int NCells {get; }

Description
Returns the number of non-empty cells.

Property Value
A int containing the number of non-empty cells.
Methods

**GetListCells**

```
public double[] GetListCells()
```

**Description**

Returns for each row, a list of the levels of nkeys corresponding classification variables that describe a cell.

**Returns**

A double array containing the list of levels of nkeys corresponding classification variables that describe a cell.

**GetTable**

```
public double[] GetTable()
```

**Description**

Returns the frequency for each cell.

**Returns**

A double array containing the frequency for each cell.
Chapter 13: Regression

Types

class RegressorsForGLM ................................................................. 625
enumeration RegressorsForGLM.DummyType ....................................... 633
class LinearRegression .............................................................. 634
class LinearRegression.CaseStatistics .............................................. 643
class LinearRegression.CoefficientTTestsValue .................................. 647
class NonlinearRegression .......................................................... 648
interface NonlinearRegression.IDerivative ...................................... 662
interface NonlinearRegression.IFunction ....................................... 663
class SelectionRegression .......................................................... 664
enumeration SelectionRegression.Criterion ....................................... 675
class SelectionRegression.SummaryStatistics .................................... 675
class StepwiseRegression ........................................................... 677
class StepwiseRegression.CoefficientTTestsValue ................................ 687
interface StepwiseRegression.Direction ......................................... 688
class UserBasisRegression .......................................................... 689
interface IRegressionBasis .......................................................... 694

Usage Notes

The regression models in this chapter include the simple and multiple linear regression models, the multivariate general linear model, and the nonlinear regression model. Functions for fitting regression models, computing summary statistics from a fitted regression, computing diagnostics, and computing confidence intervals for individual cases are provided. This chapter also provides methods for building a model from a set of candidate variables.

Simple and Multiple Linear Regression

The simple linear regression model is

\[ y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad i = 1, 2, \ldots, n \]

where the observed values of the \( y_i \)'s constitute the responses or values of the dependent variable, the \( x_i \)'s
are the settings of the independent (explanatory) variable, $\beta_0$ and $\beta_1$ are the intercept and slope parameters (respectively) and the $\varepsilon_i$'s are independently distributed normal errors, each with mean 0 and variance $\sigma^2$.

The multiple linear regression model is

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_k x_{ki} + \varepsilon_i \quad i = 1, 2, \ldots, n$$

where the observed values of the $y_i$'s constitute the responses or values of the dependent variable; the $x_{1i}, x_{2i}, \ldots, x_{ki}$'s are the settings of the $k$ independent (explanatory) variables; $\beta_0, \beta_1, \ldots, \beta_k$ are the regression coefficients; and the $\varepsilon_i$'s are independently distributed normal errors, each with mean 0 and variance $\sigma^2$.

The class `LinearRegression` fits both the simple and multiple linear regression models using a fast Given’s transformation and includes an option for excluding the intercept $\beta_0$. The responses are input in array $y$, and the independent variables are input in array $x$, where the individual cases correspond to the rows and the variables correspond to the columns.

After the model has been fitted using the `LinearRegression` class, properties such as `CoefficientTTests` can be used to retrieve summary statistics. Predicted values, confidence intervals, and case statistics for the fitted model can be obtained from inner class `LinearRegression.CaseStatistics`.

### No Intercept Model

Several functions provide the option for excluding the intercept from a model. In most practical applications, the intercept should be included in the model. For functions that use the sums of squares and crossproducts matrix as input, the no-intercept case can be handled by using the raw sums of squares and crossproducts matrix as input in place of the corrected sums of squares and crossproducts. The raw sums of squares and crossproducts matrix can be computed as $(x_1, x_2, \ldots, x_k, y)^T (x_1, x_2, \ldots, x_k, y)$.

### Specification of X for the General Linear Model

Variables used in the general linear model are either continuous or classification variables. Typically, multiple regression models use continuous variables, whereas analysis of variance models use classification variables. Although the notation used to specify analysis of variance models and multiple regression models may look quite different, the models are essentially the same. The term *general linear model* emphasizes that a common notational scheme is used for specifying a model that may contain both continuous and classification variables.

A general linear model is specified by its effects (sources of variation). An effect is referred to in this text as a single variable or a product of variables. (The term *effect* is often used in a narrower sense, referring only to a single regression coefficient.) In particular, an effect is composed of one of the following:

1. a single continuous variable
2. a single classification variable
3. several different classification variables
4. several continuous variables, some of which may be the same
5. continuous variables, some of which may be the same, and classification variables, which must be distinct

Effects of the first type are common in multiple regression models. Effects of the second type appear as main effects in analysis of variance models. Effects of the third type appear as interactions in analysis of variance models. Effects of the fourth type appear in polynomial models and response surface models as powers and crossproducts of some basic variables. Effects of the fifth type appear in one-way analysis of covariance models as regression coefficients that indicate lack of parallelism of a regression function across the groups.

The analysis of a general linear model occurs in two stages. The first stage calls class 
RegressorsForGLM to specify all regressors except the intercept. The second stage uses 
LinearRegression, at which point the model will be specified as either having or not having an intercept.

Suppose the data matrix has as its first four columns two continuous variables in Columns 0 and 1 and two classification variables in Columns 2 and 3. The data might appear as follows:

<table>
<thead>
<tr>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.23</td>
<td>1.23</td>
<td>1.0</td>
<td>5.0</td>
</tr>
<tr>
<td>12.12</td>
<td>2.34</td>
<td>1.0</td>
<td>4.0</td>
</tr>
<tr>
<td>12.34</td>
<td>1.23</td>
<td>1.0</td>
<td>4.0</td>
</tr>
<tr>
<td>4.34</td>
<td>2.21</td>
<td>1.0</td>
<td>5.0</td>
</tr>
<tr>
<td>5.67</td>
<td>4.31</td>
<td>2.0</td>
<td>4.0</td>
</tr>
<tr>
<td>4.12</td>
<td>5.34</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>4.89</td>
<td>9.31</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>9.12</td>
<td>3.71</td>
<td>2.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Each distinct value of a classification variable determines a level. The classification variable in Column 2 has two levels. The classification variable in Column 3 has three levels. (Integer values are recommended, but not required, for values of the classification variables. The values of the classification variables corresponding to the same level must be identical.) Some examples of regression functions and their specifications are as follows:

- \[ \beta_0 + \beta_1 x_1 \]  
  - Intercept: true  
  - Class Columns: \{\}\n
- \[ \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 \]  
  - Intercept: true  
  - Class Columns: \{\}\n
- \[ \mu + \alpha_i \]  
  - Intercept: true  
  - Class Columns: \{2\}\n
- \[ \mu + \alpha_i + \beta_j + \gamma_j \]  
  - Intercept: true  
  - Class Columns: \{2, 3\}\n
- \[ \mu_{ij} \]  
  - Intercept: false  
  - Class Columns: \{2, 3\}\n
- \[ \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 \]  
  - Intercept: true  
  - Class Columns: \{\}\n
Regression • 619
Variable Selection

Variable selection can be performed by `SelectionRegression`, which computes all best-subset regressions, or by `StepwiseRegression`, which computes stepwise regression. The method used by `SelectionRegression` is generally preferred over that used by `StepwiseRegression` because `SelectionRegression` implicitly examines all possible models in the search for a model that optimizes some criterion while stepwise does not examine all possible models. However, the computer time and memory requirements for `SelectionRegression` can be much greater than that for `StepwiseRegression` when the number of candidate variables is large.

Nonlinear Regression Model

The nonlinear regression model is

\[ y_i = f(x_i; \theta) + \epsilon_i \]

where the observed values of the \( y_i \)'s constitute the responses or values of the dependent variable, the \( x_i \)'s are the known vectors of values of the independent (explanatory) variables, \( f \) is a known function of an unknown regression parameter vector \( \theta \), and the \( \epsilon_i \)'s are independently distributed normal errors each with mean 0 and variance \( \sigma^2 \).

Class `NonlinearRegression` performs the least-squares fit to the data for this model.

Weighted Least Squares

Classes throughout the chapter generally allow weights to be assigned to the observations. A `weight` argument is used throughout to specify the weighting for particular rows of \( X \).

Computations that relate to statistical inference—e.g., \( t \) tests, \( F \) tests, and confidence intervals—are based on the multiple regression model except that the variance of \( \epsilon_i \) is assumed to equal \( \sigma^2 \) times the reciprocal of the corresponding weight.

If a single row of the data matrix corresponds to \( n_i \) observations, the vector `frequencies` can be used to specify the frequency for each row of \( X \). Degrees of freedom for error are affected by frequencies but are unaffected by weights.

Summary Statistics

Property and methods `LinearRegression.ANOVA`, `LinearRegression.CoefficientTTests`, `NonlinearRegression.GetR()` and `StepwiseRegression.CoefficientVIF` can be used to compute statistics related to a regression for each of the dependent variables fitted by the indicated
regression. The summary statistics include the model analysis of variance table, sequential sums of squares and \( F \)-statistics, coefficient estimates, estimated standard errors, t-statistics, variance inflation factors and estimated variance-covariance matrix of the estimated regression coefficients.

The summary statistics are computed under the model \( y = X \beta + \epsilon \), where \( y \) is the \( n \times 1 \) vector of responses, \( X \) is the \( n \times p \) matrix of regressors with rank \( (X) = r \), is the \( p \times 1 \) vector of regression coefficients, and \( \epsilon \) is the \( n \times 1 \) vector of errors whose elements are independently normally distributed with mean 0 and variance \( \sigma^2/w_i \).

Given the results of a weighted least-squares fit of this model (with the \( w_i \)’s as the weights), most of the computed summary statistics are output in the following variables:

ANOVA Class

The ANOVA property in several of the regression classes returns an ANOVA object. Summary statistics can be retrieved via specific “get” methods or the \( \text{ANOVA.GetArray()} \) method. This returns a one-dimensional array. In \( \text{StepwiseRegression} \), \( \text{ANOVA.GetArray()} \) returns \( \text{Double.NaN} \) for the last two elements of the array because they cannot be computed from the input. The array contains statistics related to the analysis of variance. The sources of variation examined are the regression, error, and total. The first 10 elements of the \( \text{ANOVA.GetArray()} \) and the notation frequently used for these is described in the following table (here, \( \text{AOV} = \text{ANOVA.GetArray()} \)):

<table>
<thead>
<tr>
<th>Variation Src.</th>
<th>Deg. of Freedom</th>
<th>Sum of Squares</th>
<th>Mean Square</th>
<th>( F )</th>
<th>( p )-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>DFE = AOV[1]</td>
<td>SSE = AOV[4]</td>
<td>( s^2 = \text{AOV[7]} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If the model has an intercept (default), the total sum of squares is the sum of squares of the deviations of \( y_i \) from its (weighted) mean \( \bar{y} \)—the so-called corrected total sum of squares, denoted by the following:

\[
\text{SST} = \sum_{i=1}^{n} w_i (y_i - \bar{y})^2
\]

If the model does not have an intercept (\( \text{hasIntercept} = \text{false} \)), the total sum of squares is the sum of squares of \( y_i \)—the so-called uncorrected total sum of squares, denoted by the following:

\[
\text{SST} = \sum_{i=1}^{n} w_i y_i^2
\]

The error sum of squares is given as follows:

\[
\text{SSE} = \sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2
\]

The error degrees of freedom is defined by \( \text{DFE} = n - r \).
The estimate of $\sigma^2$ is given by $s^2 = \frac{SSE}{DFE}$, which is the error mean square.

The computed $F$ statistic for the null hypothesis, $H_0 : \beta_1 = \beta_2 = \ldots = \beta_k = 0$, versus the alternative that at least one coefficient is nonzero is given by $F = \frac{s^2}{MSR}$. The p-value associated with the test is the probability of a larger $F$ under the assumption of the model and the null hypothesis. A small $p$-value (less than 0.05) is customarily used to indicate there is sufficient evidence from the data to reject the null hypothesis.

The remaining five elements in AOV frequently are displayed together with the actual analysis of variance table. The quantities $R$-squared ($R^2 = AOV[10]$) and adjusted $R$-squared

$$R_a^2 = (AOV[11])$$

are expressed as a percentage and are defined as follows:

$$R^2 = 100 \left(\frac{SSR}{SST}\right) = 100 \left(1 - \frac{SSE}{SST}\right)$$

$$R_a^2 = 100 \max \left\{0, 1 - \frac{s^2}{SST/DFT} \right\}$$

The square root of $s^2$ ($s = AOV[12]$) is frequently referred to as the estimated standard deviation of the model error.

The overall mean of the responses $\bar{y}$ is output in AOV[13].

The coefficient of variation ($CV = AOV[14]$) is expressed as a percentage and defined by $CV = 100s/\bar{y}$.

LinearRegression.CoefficientTTests

A nested class within the LinearRegression and StepwiseRegression classes. The statistics (estimated standard error, $t$ statistic and $p$-value) associated with each coefficient can be retrieved via associated “Get” methods.

GetR()

Estimated variance-covariance matrix of the estimated regression coefficients.

Diagnostics for Individual Cases

Diagnostics for individual cases (observations) are computed by the LinearRegression.CaseStatistics class for linear regression.

Statistics computed include predicted values, confidence intervals, and diagnostics for detecting outliers and cases that greatly influence the fitted regression.

The diagnostics are computed under the model $y = X\beta + \varepsilon$, where $y$ is the $n \times 1$ vector of responses, $X$ is the $n \times p$ matrix of regressors with $\text{rank}(X) = r$, $\beta$ is the $p \times 1$ vector of regression coefficients, and $\varepsilon$ is the $n \times 1$ vector of errors whose elements are independently normally distributed with mean 0 and variance $\phi^2/w_i$.

Given the results of a weighted least-squares fit of this model (with the $w_i$'s as the weights), the following five diagnostics are computed:
1. leverage
2. standardized residual
3. jackknife residual
4. Cook’s distance
5. DFFITS

The definition of these terms is given in the discussion that follows: Let $x_i$ be a column vector containing the elements of the $i$-th row of $X$. A case can be unusual either because of $x_i$ or because of the response $y_i$. The leverage $h_i$ is a measure of uniqueness of the $x_i$. The leverage is defined by

$$h_i = [x_i^T (X^T W X)^{-1} x_i] w_i$$

where $W = \text{diag}(w_1, w_2, \ldots, w_n)$ and $(X^T W T)^{-1}$ denotes a generalized inverse of $X^T W T$. The average value of the $h_i$’s is $r/n$. Regression functions declare $x_i$ unusual if $h_i > 2r/n$. Hoaglin and Welsch (1978) call a data point highly influential (i.e., a leverage point) when this occurs.

Let $e_i$ denote the residual

$$y_i - \hat{y}_i$$

for the $i$-th case. The estimated variance of $e_i$ is $(1 - h_i)s^2w_i$, where $s^2$ is the residual mean square from the fitted regression. The $i$-th standardized residual (also called the internally studentized residual) is by definition

$$r_i = e_i \sqrt{\frac{w_i}{s^2 (1 - h_i)}}$$

and $r_i$ follows an approximate standard normal distribution in large samples.

The $i$-th jackknife residual or deleted residual involves the difference between $y_i$ and its predicted value, based on the data set in which the $i$-th case is deleted. This difference equals $e_i/(1 - h_i)$. The jackknife residual is obtained by standardizing this difference. The residual mean square for the regression in which the $i$-th case is deleted is as follows:

$$s^2_i = \frac{(n - r)s^2 - w_i e_i^2 / (1 - h_i)}{n - r - 1}$$

The jackknife residual is defined as

$$t_i = e_i \sqrt{\frac{w_i}{s^2_i (1 - h_i)}}$$

and $t_i$ follows a $t_i$ distribution with $n - r \times 1$ degrees of freedom.
Cook's distance for the $i$-th case is a measure of how much an individual case affects the estimated regression coefficients. It is given as follows:

$$D_i = \frac{w_i h_i e_i^2}{s^2 (1 - h_i)^2}$$

Weisberg (1985) states that if $D_i$ exceeds the 50-th percentile of the $F(r, n - r)$ distribution, it should be considered large. (This value is about 1. This statistic does not have an $F$ distribution.)

DFFITS, like Cook’s distance, is also a measure of influence. For the $i$-th case, DFFITS is computed by the formula below.

$$\text{DFFITS}_i = e_i \sqrt{\frac{w_i h_i}{s^2_i (1 - h_i)^2}}$$

Hoaglin and Welsch (1978) suggest that DFFITS greater than

$$2\sqrt{r/n}$$

is large.

**Transformations**

Transformations of the independent variables are sometimes useful in order to satisfy the regression model. The inclusion of squares and crossproducts of the variables

$$(x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

is often needed. Logarithms of the independent variables are used also. (See Draper and Smith 1981, pp. 218-222; Box and Tidwell 1962; Atkinson 1985, pp. 177-180; Cook and Weisberg 1982, pp. 78-86.)

When the responses are described by a nonlinear function of the parameters, a transformation of the model equation often can be selected so that the transformed model is linear in the regression parameters. For example, by taking natural logarithms on both sides of the equation, the exponential model

$$y = e^{\beta_0 + \beta_1 x_1}$$

can be transformed to a model that satisfies the linear regression model provided the $\epsilon_i$’s have a log-normal distribution (Draper and Smith, pp. 222-225).

When the responses are nonnormal and their distribution is known, a transformation of the responses can often be selected so that the transformed responses closely satisfy the regression model, assumptions. The square-root transformation for counts with a Poisson distribution and the arc-sine transformation for binomial proportions are common examples (Snedecor and Cochran 1967, pp. 325-330; Draper and Smith, pp. 237-239).
Missing Values

NaN (Not a Number) is the missing value code used by the regression functions. Use field Double.NaN to retrieve NaN. Any element of the data matrix that is missing must be set to Double.NaN. In fitting regression models, any observation containing NaN for the independent, dependent, weight, or frequency variables is omitted from the computation of the regression parameters.

RegressorsForGLM Class

```java
public class Imsl.Stat.RegressorsForGLM
```

Generates regressors for a general linear model.

Class RegressorsForGLM generates regressors for a general linear model from a data matrix. The data matrix can contain classification variables as well as continuous variables. Regressors for effects composed solely of continuous variables are generated as powers and crossproducts. Consider a data matrix containing continuous variables as Columns 3 and 4. The effect indices (3, 3) generate a regressor whose \( i \)-th value is the square of the \( i \)-th value in Column 3. The effect indices (3, 4) generates a regressor whose \( i \)-th value is the product of the \( i \)-th value in Column 3 with the \( i \)-th value in Column 4.

Regressors for an effect (source of variation) composed of a single classification variable are generated using indicator variables. Let the classification variable \( A \) take on values \( a_1, a_2, \ldots, a_n \). From this classification variable, RegressorsForGLM creates \( n \) indicator variables. For \( k = 1, 2, \ldots, n \), we have

\[
I_k = \begin{cases} 
1 & \text{if } A = a_k \\
0 & \text{otherwise}
\end{cases}
\]

For each classification variable, another set of variables is created from the indicator variables. These new variables are called dummy variables. Dummy variables are generated from the indicator variables in one of three manners:

1. The dummies are the \( n \) indicator variables.
2. The dummies are the first \( n - 1 \) indicator variables.
3. The \( n - 1 \) dummies are defined in terms of the indicator variables so that for balanced data, the usual summation restrictions are imposed on the regression coefficients.

In particular, for dummy method All, the dummy variables are \( A_k = I_k \ (k = 1, 2, \ldots, n) \). For dummy method LeaveOutLast, the dummy variables are \( A_k = I_k \ (k = 1, 2, \ldots, n - 1) \). For dummy method SumToZero, the dummy variables are \( A_k = I_k - I_n \ (k = 1, 2, \ldots, n - 1) \). The regressors generated for an effect composed of a single-classification variable are the associated dummy variables.

Let \( m_j \) be the number of dummies generated for the \( j \)-th classification variable. Suppose there are two classification variables \( A \) and \( B \) with dummies

\[
A_1, A_2, \ldots, A_{m_1}
\]

\[
B_1, B_2, \ldots, B_{m_2}
\]
and

\[ B_1, B_2, \ldots, B_m \]

The regressors generated for an effect composed of two classification variables \( A \) and \( B \) are

\[
A \otimes B = (A_1, A_2, \ldots, A_{m_1}) \otimes (B_1, B_2, \ldots, B_{m_2})
\]

\[
= (A_1 B_1, A_1 B_2, \ldots, A_1 B_{m_2}, A_2 B_1, A_2 B_2, \ldots, A_2 B_{m_2})
\]

More generally, the regressors generated for an effect composed of several classification variables and several continuous variables are given by the Kronecker products of variables, where the order of the variables is specified in \( \text{SetEffects} \). Consider a data matrix containing classification variables in Columns 0 and 1 and continuous variables in Columns 2 and 3. Label these four columns \( A, B, X_1, \) and \( X_2 \). The regressors generated by the effect indices \( \{0, 1, 2, 3\} \) are \( A \otimes B \otimes X_1 X_2 \).

**Remarks**

Let the data matrix \( x = (A, B, X_1) \), where \( A \) and \( B \) are classification variables and \( X_1 \) is a continuous variable. The model containing the effects \( A, B, AB, X_1, AX_1, BX_1, \) and \( ABX_1 \) is specified by setting \( \text{nClassVariables} = 2 \) in the constructor and calling \( \text{SetEffects(effects)} \), with int \( \text{effects[][]} = \{ \{0\}, \{1\}, \{0, 1\}, \{2\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\} \} \);

For this model, suppose that variable \( A \) has two levels, \( A_1 \) and \( A_2 \), and that variable \( B \) has three levels, \( B_1, B_2, \) and \( B_3 \). For each \( \text{DummyMethod} \) option, the regressors in their order of appearance in regressors are given below.

<table>
<thead>
<tr>
<th>DummyMethod</th>
<th>Regressors</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>( A_1, A_2, B_1, B_2, B_3, A_1 B_1, A_1 B_2, A_2 B_1, A_2 B_2, A_2 B_3, X_1, A_1 X_1, A_1 X_2, A_2 X_1, A_2 X_2, A_1 B_1 X_1, A_2 B_2 X_1, A_3 B_3 X_1 )</td>
</tr>
<tr>
<td>LeaveOutLast</td>
<td>( A_1, B_1, B_2, A_1 B_1, A_1 B_2, X_1, A_1 X_1, B_1 X_1, B_2 X_1, A_1 B_1 X_1, A_1 B_2 X_1, A_2 B_3 X_1 )</td>
</tr>
<tr>
<td>SumToZero</td>
<td>( A_1 - A_2, B_1 - B_2, B_2 - B_3, (A_1 - A_2)(B_1 - B_2), (A_1 - A_2)(B_2 - B_3), X_1, (A_1 - A_2)X_1, (B_1 - B_3)X_1, (B_2 - B_3)X_1, (A_1 - A_2)(B_1 - B_2)X_1, (A_1 - A_2)(B_2 - B_3)X_1 )</td>
</tr>
</tbody>
</table>

Within a group of regressors corresponding to an interaction effect, the indicator variables composing the regressors vary most rapidly for the last classification variable, next most rapidly for the next to last classification variable, etc.

By default, \( \text{RegressorsForGLM} \) internally generates values for effects which correspond to a first order model with \( \text{nEffects} = \text{nContinuousVariables} + \text{nClassVariables} \), where \( \text{nContinuousVariables} \) is the number of continuous variables and \( \text{nClassVariables} \) is the number of classification variables. The variables then are used to create the regressor variables. The effects are ordered such that the first effect corresponds to the first column of \( x \), the second effect corresponds to the second column of \( x \), etc. A second order model corresponding to the columns (variables) of \( x \) is generated if \( \text{ModelOrder} = 2 \) is used.
The effects array for a first or second order model can be obtained by first using **ModelOrder** followed by **GetEffects**. This array can then be modified and used as the argument to **SetEffects**. This may be an easier way of setting the effects for an almost linear or quadratic model than creating the effects array from scratch.

There are

\[ nEffects = n\text{ClassVariables} + n\text{ContinuousVariables} + \frac{n\text{Var}(n\text{Var} - 1)}{2} \]

effects, where \( n\text{Var} = n\text{ClassVariables} + n\text{ContinuousVariables} \). The first \( n\text{Var} \) effects correspond to the columns of \( x \), such that the first effect corresponds to the first column of \( x \), the second effect corresponds to the second column of \( x \), ..., the \( n\text{Var} \)-th effect corresponds to the \( n\text{Var} \)-th column of \( x \) (i.e. \( x[n\text{Var}-1] \)). The next \( n\text{ContinuousVariables} \) effects correspond to squares of the continuous variables. The last \( n\text{Var}(n\text{Var} - 1)/2 \) effects correspond to the two-variable interactions.

- Let the data matrix \( x = (A, B, X_1) \), where \( A \) and \( B \) are classification variables and \( X_1 \) is a continuous variable. The effects generated and order of appearance is
  
  \[ A, B, X_1, X_1^2, AB, AX_1, BX_1 \]

- Let the data matrix \( x = (A, X_1, X_2) \), where \( A \) is a classification variable and \( X_1 \) and \( X_2 \) are continuous variables. The effects generated and order of appearance is
  
  \[ A, X_1, X_2, X_1^2, X_2^2, AX_1, AX_2, X_1X_2 \]

- Let the data matrix \( x = (X_1, A, X_2) \), where \( A \) is a classification variable and \( X_1 \) and \( X_2 \) are continuous variables. The effects generated and order of appearance is
  
  \[ X_1, A, X_2, X_1^2, X_2^2, X_1A, X_1X_2, AX_2 \]

Higher-order and more complicated models can be specified using **SetEffects**.

### Properties

**DummyMethod**

```csharp
cpyublic Imsl.Stat.RegressorsForGLM.DummyType DummyMethod {get; set; }
```

**Description**

The dummy method.

**Remarks**

One of **All** (the default), **LeaveOutLast** or **SumToZero**.

**ModelOrder**

```csharp
public int ModelOrder {set; }
```

### Regression
Description
The order of the model.

Remarks
Model order can be specified as 1 or 2. Use SetEffects to specify more complicated models. This overrides previously set effects.

NumberOfMissingRows
public int NumberOfMissingRows {get; }

Description
Returns the number of rows in the regressors matrix containing NaN (not a number).

Remarks
A row of the regressors matrix contains NaN for a regressor when any of the variables involved in generation of the regressor equals NaN.

NumberOfRegressors
public int NumberOfRegressors {get; }

Description
Returns the number of regressors.

 Constructors

RegressorsForGLM
public RegressorsForGLM(double[,] x, int nClassVariables)

Description
Constructor where the class columns are the first columns.

Parameters
- x – is an nObservations by nClassVariables+nContinuousVariables array containing the data, where nObservations is the number of observations. The columns must be ordered such that the first nClassVariables columns contain the class variables and the next nContinuousVariables columns contain the continuous variables.
- nClassVariables – is number of class variables. The number of continuous variables is assumed to be the number of columns in x-nClassVariables.

RegressorsForGLM
public RegressorsForGLM(double[,] x, int[] classColumns)

Description
Constructor with an explicit set of class column indices.
**Parameters**

- **x** – is an nObservations by nClassVariables+nContinuousVariables array containing the data. The columns containing the class variables are specified by **classColumns**.
- **classColumns** – is an array containing the column indices, in x, of the class variables.

**Methods**

**GetEffects**

```csharp
public int[][] GetEffects()
```

**Description**
Returns the effects.

**Returns**
a jagged array containing the effects. The number of rows in the matrix is the number of effects. For each row, the values are the 0-based column numbers of x.

**GetEffectsColumns**

```csharp
public int[][] GetEffectsColumns()
```

**Description**
Returns a mapping of effects to regressor columns.

**Returns**
A jagged int array. The number of rows is equal to the number of effects. Each row contains the column numbers of the regressor matrix into which the corresponding effect is mapped.

**GetRegressors**

```csharp
public double[,] GetRegressors()
```

**Description**
Returns the regressor array.

**Returns**
An array of size number of observations by number of regressors.

**SetEffects**

```csharp
public void SetEffects(int[][] effects)
```

**Description**
Set the effects.

**Parameter**
- **effects** – A jagged array containing the effects. The number of rows in the matrix is the number of effects. For each row, the values are the 0-based column numbers of x.
Remarks
This overrides any previously set model order. Effects is a jagged array. The number of rows in the matrix is the number of effects. For each row, the values are the 0-based column numbers of x.

Example 1
In the following example, there are two classification variables, A and B, with two and three values, respectively. Regressors for a one-way model (the default model order) are generated using the the default dummy method. The five regressors generated are $A_1, A_2, B_1, B_2$ and $B_3$.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;

public class RegressorsForGLMEx1
{
    public static void Main(String[] args)
    {
        double[,] x = {
            {10.0, 5.0},
            {20.0, 15.0},
            {20.0, 10.0},
            {10.0, 10.0},
            {10.0, 15.0},
            {20.0, 5.0}
        };

        RegressorsForGLM r = new RegressorsForGLM(x, 2);
        double[,] regressors = r.GetRegressors();
        int n = r.NumberOfRegressors;
        Console.WriteLine("Number of regressors = "+n);
        Console.WriteLine();
        new PrintMatrix("Regressors").Print(regressors);
    }
}
```

Output
Number of regressors = 5

<table>
<thead>
<tr>
<th>Regressors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
</tr>
<tr>
<td>0 1 0 1 0 0</td>
</tr>
<tr>
<td>1 0 1 0 0 1</td>
</tr>
<tr>
<td>2 0 1 0 1 0</td>
</tr>
<tr>
<td>3 1 0 0 1 0</td>
</tr>
<tr>
<td>4 1 0 0 0 1</td>
</tr>
<tr>
<td>5 0 1 1 0 0</td>
</tr>
</tbody>
</table>
Example 2

In this example, a two-way analysis of covariance model containing all the interaction terms is fit. First, `RegressorsForGLM` is used to produce a matrix of regressors, `regressors`, from the data `x`. Then, `regressors` is used as the input matrix into `Regression` to produce the final fit. The regressors, generated using the dummy method `LeaveOutLast`, are the model whose mean function is

\[
\mu + \alpha_i + \beta_j + \gamma_{ij} + \delta_{ij}x + \xi_{ij}x + \eta_{ij} + \theta_{ij}x_{ij} \quad i = 1, 2; \quad j = 1, 2, 3
\]

where \( \alpha_2 = \beta_3 = \gamma_{21} + \gamma_{22} + \gamma_{23} + \xi_{2} + \eta_{3} + \theta_{21} + \theta_{22} + \theta_{23} = 0. \)

```java
using System;
using Imsl.Stat;
using Imsl.Math;

class RegressorsForEx2
{
    public static void Main(String[] args)
    {
        double[,] x = {
            {1.0, 1.0, 1.11},
            {1.0, 1.0, 2.22},
            {1.0, 1.0, 3.33},
            {1.0, 2.0, 1.11},
            {1.0, 2.0, 2.22},
            {1.0, 2.0, 3.33},
            {1.0, 3.0, 1.11},
            {1.0, 3.0, 2.22},
            {1.0, 3.0, 3.33},
            {2.0, 1.0, 1.11},
            {2.0, 1.0, 2.22},
            {2.0, 1.0, 3.33},
            {2.0, 2.0, 1.11},
            {2.0, 2.0, 2.22},
            {2.0, 2.0, 3.33},
            {2.0, 3.0, 1.11},
            {2.0, 3.0, 2.22},
            {2.0, 3.0, 3.33}
        };
        double[] y = {
            1.0, 2.0, 2.0, 4.0, 4.0, 6.0,
            3.0, 3.5, 4.0, 4.5, 5.0, 5.5,
            2.0, 3.0, 4.0, 5.0, 6.0, 7.0
        };
        int[][] effects = {
            new int[]{0},
            new int[]{1},
            new int[]{0, 1},
            new int[]{2},
            new int[]{0, 2},
            new int[]{1, 2},
            new int[]{0, 1, 2}
        };
    }
}
```
int nClassVariables = 2;

RegressorsForGLM r = new RegressorsForGLM(x, nClassVariables);
r.DummyMethod = RegressorsForGLM.DummyType.LeaveOutLast;
r.SetEffects(effects);

int nRegressors = r.NumberOfRegressors;

double[,] regressors = r.GetRegressors();


new Imsl.Math.PrintMatrix("Regressors").Print(pmf, regressors);

LinearRegression regression = new LinearRegression(nRegressors, true);
regression.Update(regressors, y);

Console.WriteLine(" * * * Analysis of Variance * * *");
ANOVA anova = regression.ANOVA;
Object[,] table = new Object[15,2];
table[0,0] = "Degrees of freedom for the model ";
table[0,1] = anova.DegreesOfFreedomForModel;
table[1,0] = "Degrees of freedom for the error ";
table[1,1] = anova.DegreesOfFreedomForError;
table[2,0] = "Total degrees of freedom ";
table[2,1] = anova.TotalDegreesOfFreedom;
table[3,0] = "Sum of squares for the model ";
table[3,1] = anova.SumOfSquaresForModel;
table[4,0] = "Sum of squares for error ";
table[4,1] = anova.SumOfSquaresForError;
table[5,0] = "Total sum of squares ";
table[5,1] = anova.TotalSumOfSquares;
table[6,0] = "Model mean square ";
table[6,1] = anova.ModelMeanSquare;
table[7,0] = "Error mean square ";
table[7,1] = anova.ErrorMeanSquare;
table[8,0] = "F-statistic ";
table[8,1] = anova.F;
table[9,0] = "p-value ";
table[9,1] = anova.P;
table[10,0] = "R-squared ";
table[10,1] = anova.RSquared;
table[11,0] = "Adjusted R-squared ";
table[11,1] = anova.AdjustedRSquared;
table[12,0] = "Standard deviation for the model error ";
table[12,1] = anova.ModelErrorStdev;
table[13,0] = "Overall mean of y ";
table[13,1] = anova.MeanOfY;
table[14,0] = "Coefficient of variation ";
table[14,1] = anova.CoefficientOfVariation;

pmf = new Imsl.Math.PrintMatrixFormat();
pmf.SetNoColumnLabels();
pmf.SetNoRowLabels();
pmf.NumberFormat = "0.0000";
new Imsl.Math.PrintMatrix().Print(pmf, table);
Dummy variable types.

**Fields**

- **All**
  public Imsl.Stat.RegressorsForGLM.DummyType All
  
  **Description**
  The $n$ indicator variables are the dummy variables.

- **LeaveOutLast**
  public Imsl.Stat.RegressorsForGLM.DummyType LeaveOutLast
  
  **Description**
  The dummies are the first $n-1$ indicator variables.

- **SumToZero**
  public Imsl.Stat.RegressorsForGLM.DummyType SumToZero
  
  **Description**
  The $n-1$ dummies are defined in terms of the indicator variables so that for balanced data, the usual summation restrictions are imposed on the regression coefficients.

**LinearRegression Class**

public class Imsl.Stat.LinearRegression

Fits a multiple linear regression model with or without an intercept.

Fits a multiple linear regression model with or without an intercept. If the constructor argument hasIntercept is true, the multiple linear regression model is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_k x_{ik} + e_i \quad i = 1, 2, \ldots, n$$

where the observed values of the $y_i$’s constitute the responses or values of the dependent variable, the $x_{i1}$’s, $x_{i2}$’s, ..., $x_{ik}$’s are the settings of the independent variables, $\beta_0, \beta_1, \ldots, \beta_k$ are the regression coefficients, and the $e_i$’s are independently distributed normal errors each with mean zero and variance $\sigma^2/w_i$. If hasIntercept is false, $\beta_0$ is not included in the model.

LinearRegression computes estimates of the regression coefficients by minimizing the sum of squares of the deviations of the observed response $y_i$ from the fitted response

$$\hat{y}_i$$
for the observations. This minimum sum of squares (the error sum of squares) is in the ANOVA output and denoted by

\[ \text{SSE} = \sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2 \]

In addition, the total sum of squares is output in the ANOVA table. For the case, hasIntercept is true; the total sum of squares is the sum of squares of the deviations of \( y_i \) from its mean \( \bar{y} \)

--the so-called corrected total sum of squares; it is denoted by

\[ \text{SST} = \sum_{i=1}^{n} w_i (y_i - \bar{y})^2 \]

For the case hasIntercept is false, the total sum of squares is the sum of squares of \( y_i \) --the so-called uncorrected total sum of squares; it is denoted by

\[ \text{SST} = \sum_{i=1}^{n} y_i^2 \]

See Draper and Smith (1981) for a good general treatment of the multiple linear regression model, its analysis, and many examples.

In order to compute a least-squares solution, LinearRegression performs an orthogonal reduction of the matrix of regressors to upper triangular form. Givens rotations are used to reduce the matrix. This method has the advantage that the loss of accuracy resulting from forming the crossproduct matrix used in the normal equations is avoided, while not requiring the storage of the full matrix of regressors. The method is described by Lawson and Hanson, pages 207-212.

From a general linear model fitted using the \( w_i \)'s as the weights, inner class Imsl.Stat.LinearRegression.CaseStatistics (p. 643) can also compute predicted values, confidence intervals, and diagnostics for detecting outliers and cases that greatly influence the fitted regression. Let \( x_i \) be a column vector containing elements of the \( i \)-th row of \( X \). Let \( W = \text{diag}(w_1, w_2, ..., w_n) \). The leverage is defined as

\[ h_i = [x_i^T (X^T WX) - x_i] w_i \]

(In the case of linear equality restrictions on \( \beta \), the leverage is defined in terms of the reduced model.) Put \( D = \text{diag}(d_1, d_2, ..., d_k) \) with \( d_j = 1 \) if the \( j \)-th diagonal element of \( R \) is positive and 0 otherwise. The leverage is computed as \( h_i = (a^T Da) w_i \) where \( a \) is a solution to \( R^T a = x_i \). The estimated variance of

\[ \hat{y}_i = x_i^T \hat{\beta} \]

is given by \( h_i \hat{s}^2 / w_i \), where \( \hat{s}^2 = \text{SSE}/\text{DFE} \). The computation of the remainder of the case statistics follows easily from their definitions.

Let \( e_i \) denote the residual

\[ y_i - \hat{y}_i \]

for the \( i \)-th case. The estimated variance of \( e_i \) is \( (1 - h_i) s^2 / w_i \) where \( s^2 \) is the residual mean square from the fitted regression. The \( i \)-th standardized residual (also called the internally studentized residual) is by definition

\[ r_i = e_i \sqrt{\frac{w_i}{s^2 (1 - h_i)}} \]
and $r_i$ follows an approximate standard normal distribution in large samples.

The $i$th jackknife residual or deleted residual involves the difference between $y_i$ and its predicted value based on the data set in which the $i$th case is deleted. This difference equals $e_i/(1 - h_i)$. The jackknife residual is obtained by standardizing this difference. The residual mean square for the regression in which the $i$th case is deleted is

$$s_i^2 = \frac{(n-r)s^2 - w_i e_i^2/(1-h_i)}{n-r-1}$$

The jackknife residual is defined to be

$$t_i = e_i \sqrt{\frac{w_i}{s_i^2(1-h_i)}}$$

and $t_i$ follows a $t$ distribution with $n - r - 1$ degrees of freedom.

Cook’s distance for the $i$th case is a measure of how much an individual case affects the estimated regression coefficients. It is given by

$$D_i = \frac{w_i h_i e_i^2}{rs^2(1-h_i)^2}$$

Weisberg (1985) states that if $D_i$ exceeds the 50-th percentile of the $F(r, n-r)$ distribution, it should be considered large. (This value is about 1. This statistic does not have an $F$ distribution.)

DFFITS, like Cook’s distance, is also a measure of influence. For the $i$th case, DFFITS is computed by the formula

$$DFFITS_i = e_i \sqrt{\frac{w_i h_i}{s_i^2(1-h_i)^2}}$$

Hoaglin and Welsch (1978) suggest that $DFFITS_i$ greater than

$$2 \sqrt{r/n}$$

is large.

Often predicted values and confidence intervals are desired for combinations of settings of the effect variables not used in computing the regression fit. This can be accomplished using a single data matrix by including these settings of the variables as part of the data matrix and by setting the response equal to Double.NaN. LinearRegression will omit the case when performing the fit and a predicted value and confidence interval for the missing response will be computed from the given settings of the effect variables.

### Properties

**ANOVA**

```csharp
public Imsl.Stat.ANOVA ANOVA {get; }
```

**Description**

Returns an analysis of variance table and related statistics.
Property Value
An ANOVA table and related statistics.

CoefficientTTests
public Imsl.Stat.LinearRegression.CoefficientTTestsValue CoefficientTTests {get; }

Description
Returns statistics relating to the regression coefficients.

Property Value
A LinearRegression.CoefficientTTestsValue object which possesses regression statistics related to the coefficients.

HasIntercept
public bool HasIntercept {get; }

Description
A bool which indicates whether or not an intercept is in this regression model.

Property Value
A bool indicating whether or not the model includes an intercept.

Rank
public int Rank {get; }

Description
Returns the rank of the matrix.

Property Value
An int which specifies the matrix rank.

Constructor
LinearRegression
public LinearRegression(int nVariables, bool hasIntercept)

Description
Constructs a new linear regression object.

Parameters
nVariables – An int which specifies the number of regression variables.
hasIntercept – A bool which indicates whether or not an intercept is in this regression model.
Methods

GetCaseStatistics
GetCaseStatistics(double[] x, double y)

Description
Returns the case statistics for an observation.

Parameters
  x – A double array containing the independent (explanatory) variables. Its length must be equal to the number of variables set in the LinearRegression constructor.
  y – A double representing the dependent (response) variable.

Returns
The CaseStatistics for the observation.

GetCaseStatistics
GetCaseStatistics(double[] x, double y, double w)

Description
Returns the case statistics for an observation and a weight.

Parameters
  x – A double array containing the independent (explanatory) variables. Its length must be equal to the number of variables set in the constructor.
  y – A double representing the dependent (response) variable.
  w – A double representing the weight.

Returns
The CaseStatistics for the observation.

GetCaseStatistics
GetCaseStatistics(double[] x, double y, int pred)

Description
Returns the case statistics for an observation and future response count for the desired prediction interval.

Parameters
  x – A double array containing the independent (explanatory) variables. Its length must be equal to the number of variables set in the constructor.
  y – A double representing the dependent (response) variable.
  pred – An int representing the number of future responses for which the prediction interval is desired on the average of the future responses.
Returns
The CaseStatistics for the observation.

**GetCaseStatistics**

virtual public Imsl.Stat.LinearRegression.CaseStatistics GetCaseStatistics(double[] x, double y, double w, int pred)

**Description**

Returns the case statistics for an observation, weight, and future response count for the desired prediction interval.

**Parameters**

- **x** – A double array containing the independent (explanatory) variables. Its length must be equal to the number of variables set in the constructor.
- **y** – A double representing the dependent (response) variable.
- **w** – A double representing the weight.
- **pred** – An int representing the number of future responses for which the prediction interval is desired on the average of the future responses.

Returns
The CaseStatistics for the observation.

**GetCoefficients**

public double[] GetCoefficients()

**Description**

Returns the regression coefficients.

**Returns**

A double array containing the regression coefficients.

**Remarks**

If HasIntercept is false its length is equal to the number of variables. If HasIntercept is true then its length is the number of variables plus one and the 0-th entry is the value of the intercept.

If the model is not full rank, the regression coefficients are not uniquely determined. In this case, a warning is issued and a solution with all linearly dependent regressors set to zero is returned.

**GetR**

public double[,] GetR()

**Description**

Returns a copy of the R matrix.

**Returns**

A double matrix containing a copy of the R matrix.
Remarks

$R$ is the upper triangular matrix containing the $R$ matrix from a QR decomposition of the matrix of regressors.

GetRank

public int GetRank()

Description

Returns the rank of the matrix.

Returns

An int containing the rank of the matrix.

Update

public void Update(double[] x, double y)

Description

Updates the regression object with a new observation.

Parameters

- $x$ – A double array containing the independent (explanatory) variables.
- $y$ – A double representing the dependent (response) variable.

Remarks

$x$.Length must be equal to the number of variables set in the constructor.

Update

public void Update(double[] x, double y, double w)

Description

Updates the regression object with a new observation and weight.

Parameters

- $x$ – A double array containing the independent (explanatory) variables.
- $y$ – A double representing the dependent (response) variable.
- $w$ – A double representing the weight.

Remarks

$x$.Length must be equal to the number of variables set in the constructor.

Update

public void Update(double[,] x, double[] y)

Description

Updates the regression object with a new set of observations.
Parameters
x – A double matrix containing the independent (explanatory) variables.
y – A double array containing the dependent (response) variables.

Remarks
The number of rows in x must equal y.Length and the number of columns must be equal to the number of variables set in the constructor.

Update
public void Update(double[,] x, double[] y, double[] w)

Description
Updates the regression object with a new set of observations and weights.

Parameters
x – A double matrix containing the independent (explanatory) variables.
y – A double array containing the dependent (response) variables.
w – A double array representing the weights.

Remarks
The number of rows in x must equal y.Length and the number of columns must be equal to the number of variables set in the constructor.

Example: Linear Regression
The coefficients of a simple linear regression model, without an intercept, are computed.

```csharp
using System;
using Imsl.Stat;

public class LinearRegressionEx1
{
    public static void Main(String[] args)
    {
        // y = 4*x0 + 3*x1
        LinearRegression r = new LinearRegression(2, false);
        double[] c = new double[]{4, 3};
        double[] x0 = {1, 5};
        double[] x1 = {0, 2};
        double[] x2 = {-1, 4};
        r.Update(x0, 1 * c[0] + 5 * c[1]);
        r.Update(x1, 0 * c[0] + 2 * c[1]);
        r.Update(x2, -1 * c[0] + 4 * c[1]);
        double[] coef = r.GetCoefficients();
        Console.Out.WriteLine("The computed regression coefficients are {" +
            coef[0] + ", " + coef[1] + "}");
    }
}
```
The computed regression coefficients are \{4, 3\}

**Example: Linear Regression Case Statistics**

Selected case statistics of a simple linear regression model, with an intercept, are computed.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;

public class LinearRegressionEx2
{
    public static void Main(String[] args)
    {
        LinearRegression r = new LinearRegression(2, true);
        double[] y = {3, 4, 5, 7, 8, 9};
        double[,] x = {{1, 1},{1, 2},{1, 3},{1,4},{1,5},{0,6},{1,7}};
        double[,] results = new double[7,5];
        double[] confint = new double[2];
        r.Update(x, y);
        double[] xTmp = new double[2];
        for (int k=0; k<7; k++){
            xTmp[0] = x[k,0];
            xTmp[1] = x[k,1];
            LinearRegression.CaseStatistics cs = r.GetCaseStatistics(xTmp,y[k]);
            results[k,0] = cs.JackknifeResidual;
            results[k,1] = cs.CooksDistance;
            results[k,2] = cs.DFFITS;
            confint = cs.ConfidenceInterval;
            results[k,3] = confint[0];
            results[k,4] = confint[1];
        }
        PrintMatrix p = new PrintMatrix("Selected Case Statistics");
        PrintMatrixFormat mf = new PrintMatrixFormat();
        String[] labels = {"Jackknife Residual.","Cook's D","DFFITS", "[Conf. Interval], "on the Mean"];
        mf.SetColumnLabels(labels);
        mf.NumberFormat = "0.#########";
        p.Print(mf, results);
    }
}
```

**Output**

<table>
<thead>
<tr>
<th>Jackknife Residual</th>
<th>Cook’s D</th>
<th>DFFITS</th>
<th>[Conf. Interval on the Mean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  -0.343038693</td>
<td>0.044885519</td>
<td>-0.323965838</td>
<td>2.260946521  3.996196336</td>
</tr>
<tr>
<td>1  -0.327326835</td>
<td>0.018390805</td>
<td>-0.207019668</td>
<td>3.467412069  4.818302217</td>
</tr>
<tr>
<td>2  -0.337597012</td>
<td>0.011212981</td>
<td>-0.16122517</td>
<td>4.612581629  5.701704085</td>
</tr>
<tr>
<td>3  Infinity</td>
<td>0.275862069</td>
<td>Infinity</td>
<td>5.648231067  6.694626076</td>
</tr>
<tr>
<td>4  -0.417763902</td>
<td>0.023512274</td>
<td>-0.236601469</td>
<td>6.562984697  7.808443875</td>
</tr>
<tr>
<td>5  NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>6.736357974  9.263642026</td>
</tr>
</tbody>
</table>

642 • LinearRegression | IMSL C# Numerical Library
LinearRegression.CaseStatistics Class

public class Imsl.Stat.LinearRegression.CaseStatistics

Inner Class CaseStatistics allows for the computation of predicted values, confidence intervals, and diagnostics for detecting outliers and cases that greatly influence the fitted regression.

Properties

ConfidenceInterval
virtual public double[] ConfidenceInterval {get; }

Description
Returns the Confidence Interval on the mean for an observation.

Property Value
A double[2] array containing the Confidence Interval for the observation.

ConLevelMean
virtual public double ConLevelMean {set; }

Description
Sets the confidence level for two-sided interval estimates on the mean, in percent.

Property Value
A double containing the confidence level on the mean.

Remarks
By default, ConLevelMean = 0.95.

ConLevelPred
virtual public double ConLevelPred {set; }

Description
Sets the confidence level for two-sided prediction intervals, in percent.

Property Value
A double containing the confidence level.
Remarks

By default, ConLevelPred = 0.95.

CooksDistance

virtual public double CooksDistance {get; }

Description

Returns Cook’s Distance for an observation.

Property Value

A double containing Cook’s Distance for an observation.

DFFITS

virtual public double DFFITS {get; }

Description

Returns DFFITS for an observation.

Property Value

A double containing the DFFITS value for an observation.

JackknifeResidual

virtual public double JackknifeResidual {get; }

Description

Returns the Jackknife Residual for an observation.

Property Value

A double containing the Jackknife Residual for an observation.

Leverage

virtual public double Leverage {get; }

Description

Returns the Leverage for an observation.

Property Value

A double containing the Leverage for an observation.

ObservedResponse

virtual public double ObservedResponse {get; }

Description

Returns the observed response for an observation.

Property Value

A double containing the observed response for an observation.

PredictedResponse

virtual public double PredictedResponse {get; }
**Description**
Returns the predicted response for an observation.

**Property Value**
A `double` containing the predicted response for an observation.

---

**PredictionInterval**

```csharp
virtual public double[] PredictionInterval {get; }
```

**Description**
Returns the Prediction Interval for an observation.

**Property Value**
A `double[2]` array containing the Prediction Interval for the observation.

---

**Residual**

```csharp
virtual public double Residual {get; }
```

**Description**
Returns the Residual for an observation.

**Property Value**
A `double` containing the residual for an observation.

---

**StandardizedResidual**

```csharp
virtual public double StandardizedResidual {get; }
```

**Description**
Returns the Standardized Residual for an observation.

**Property Value**
A `double` containing the Standardized Residual for an observation.

---

**Statistics**

```csharp
virtual public double[] Statistics {get; }
```

**Description**
Returns the case statistics for an observation.

**Property Value**

**Remarks**
Elements 0 through 11 contain the following:

---

**Regression**

`LinearRegression.CaseStatistics` • 645
### Example: Linear Regression Case Statistics

Selected case statistics of a simple linear regression model, with an intercept, are computed.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;

public class LinearRegressionEx2
{
    public static void Main(String[] args)
    {
        LinearRegression r = new LinearRegression(2, true);
        double[] y = {3, 4, 5, 7, 8, 9};
        double[,] x = {{{1, 1},{1, 2},{1, 3},{1, 4},{1,5},{0,6},{1,7}}};
        double[,] results = new double[7,5];
        double[] confint = new double[2];
        r.Update(x, y);
        double[] xTmp = new double[2];
        for (int k=0; k<7; k++){
            xTmp[0] = x[k,0];
            xTmp[1] = x[k,1];
            LinearRegression.CaseStatistics cs = r.GetCaseStatistics(xTmp,y[k]);
            results[k,0] = cs.JackknifeResidual;
            results[k,1] = cs.CooksDistance;
            results[k,2] = cs.DFFITS;
            confint = cs.ConfidenceInterval;
            results[k,3] = confint[0];
            results[k,4] = confint[1];
        }
        PrintMatrix p = new PrintMatrix("Selected Case Statistics");
        PrintMatrixFormat mf = new PrintMatrixFormat();
        String[] labels = {"Jackknife Residual.","Cook's D","DFFITS","[Conf. Interval], "on the Mean"]};
        mf.SetColumnLabels(labels);
        mf.NumberFormat = "0.############";
        p.Print(mf, results);
    }
}
```

### Table: Case Statistics

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Observed response</td>
</tr>
<tr>
<td>1</td>
<td>Predicted response</td>
</tr>
<tr>
<td>2</td>
<td>Residual</td>
</tr>
<tr>
<td>3</td>
<td>Leverage</td>
</tr>
<tr>
<td>4</td>
<td>Standardized residual</td>
</tr>
<tr>
<td>5</td>
<td>Jackknife residual</td>
</tr>
<tr>
<td>6</td>
<td>Cook's distance</td>
</tr>
<tr>
<td>7</td>
<td>DFFITS</td>
</tr>
<tr>
<td>8,9</td>
<td>Confidence interval on the mean</td>
</tr>
<tr>
<td>10,11</td>
<td>Prediction interval</td>
</tr>
</tbody>
</table>
Output

<table>
<thead>
<tr>
<th>Jackknife Residual</th>
<th>Cook's D</th>
<th>DFFITS</th>
<th>[Conf. Interval on the Mean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.343038693</td>
<td>0.044885519</td>
<td>-0.323965838 - 2.260946521</td>
</tr>
<tr>
<td>1</td>
<td>-0.327326835</td>
<td>0.018390805</td>
<td>-0.207019668 - 3.467412069</td>
</tr>
<tr>
<td>2</td>
<td>-0.337597012</td>
<td>0.011129861</td>
<td>-0.16122517 - 4.612581629</td>
</tr>
<tr>
<td>3</td>
<td>Infinity</td>
<td>0.275862069</td>
<td>Infinity - 5.648231067 - 6.894626076</td>
</tr>
<tr>
<td>4</td>
<td>-0.417763902</td>
<td>0.023512274</td>
<td>-0.236601469 - 6.562984697</td>
</tr>
<tr>
<td>5</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN - 6.736357974 - 9.263642026</td>
</tr>
<tr>
<td>6</td>
<td>-0.742307489</td>
<td>0.372413793</td>
<td>-0.995910003 - 8.201118103 - 10.227453326</td>
</tr>
</tbody>
</table>

**LinearRegression.CoefficientTTestsValue Class**

public class Imsl.Stat.LinearRegression.CoefficientTTestsValue

CoefficientTTestsValue contains statistics related to the regression coefficients.

**Constructor**

CoefficientTTestsValue

public CoefficientTTestsValue(Imsl.Stat.LinearRegression lr)

**Description**

CoefficientTTestsValue contains statistics related to the regression coefficients.

**Parameter**

lr – A LinearRegression object used to calculate the regression statistics.

**Methods**

**GetCoefficient**

public double GetCoefficient(int i)

**Description**

Returns the estimate for a coefficient.

**Parameter**

i – An int which specifies the index of the coefficient whose estimate is to be returned.
Returns
A double which specifies the estimate for the \( i \)-th coefficient.

GetPValue
public double GetPValue(int i)

Description
Returns the p-value for the two-sided test.

Parameter
\( i \) – An int which specifies the index of the coefficient whose \( p \)-value estimate is to be returned.

Returns
A double which specifies the estimated \( p \)-value for the \( i \)-th coefficient estimate.

GetStandardError
public double GetStandardError(int i)

Description
Returns the estimated standard error for a coefficient estimate.

Parameter
\( i \) – An int which specifies the index of the coefficient whose standard error estimate is to be returned.

Returns
A double which specifies the estimated standard error for the \( i \)-th coefficient estimate.

GetTStatistic
public double GetTStatistic(int i)

Description
Returns the t-statistic for the test that the \( i \)-th coefficient is zero.

Parameter
\( i \) – An int which specifies the index of the coefficient whose standard error estimate is to be returned.

Returns
A double which specifies the estimated standard error for the \( i \)-th coefficient estimate.

NonlinearRegression Class

public class Imsl.Stat.NonlinearRegression

648 • NonlinearRegression

IMSL C# Numerical Library
Fits a multivariate nonlinear regression model using least squares.

The nonlinear regression model is

$$y_i = f(x_i; \theta) + \varepsilon_i \quad i = 1, 2, \ldots, n$$

where the observed values of the $y_i$ constitute the responses or values of the dependent variable, the known $x_i$ are vectors of values of the independent (explanatory) variables, $\theta$ is the vector of $p$ regression parameters, and the $\varepsilon_i$ are independently distributed normal errors each with mean zero and variance $\sigma^2$.

For this model, a least squares estimate of $\theta$ is also a maximum likelihood estimate of $\theta$.

The residuals for the model are

$$e_i(\theta) = y_i - f(x_i; \theta) \quad i = 1, 2, \ldots, n$$

A value of $\theta$ that minimizes

$$\sum_{i=1}^n |e_i(\theta)|^2$$

is the least-squares estimate of $\theta$ calculated by this class. NonlinearRegression accepts these residuals one at a time as input from a user-supplied function. This allows NonlinearRegression to handle cases where $n$ is so large that data cannot reside in an array but must reside in a secondary storage device.

NonlinearRegression is based on MINPACK routines LMDIF and LMDER by More’ et al. (1980). NonlinearRegression uses a modified Levenberg-Marquardt method to generate a sequence of approximations to the solution. Let $\hat{\theta}_c$ be the current estimate of $\theta$. A new estimate is given by

$$\hat{\theta}_c + s_c$$

where $s_c$ is a solution to

$$(J(\hat{\theta}_c)^T J(\hat{\theta}_c) + \mu_c I)s_c = J(\hat{\theta}_c)^T e(\hat{\theta}_c)$$

Here, $J(\hat{\theta}_c)$ is the Jacobian evaluated at $\hat{\theta}_c$.

The algorithm uses a “trust region” approach with a step bound of $\hat{\delta}_c$. A solution of the equations is first obtained for $\mu_c = 0$. If $|s_c|_2 < \hat{\delta}_c$, this update is accepted; otherwise, $\mu_c$ is set to a positive value and another solution is obtained. The method is discussed by Levenberg (1944), Marquardt (1963), and Dennis and Schnabel (1983, pages 129 - 147, 218 - 338).

Forward finite differences are used to estimate the Jacobian numerically unless the user supplied function computes the derivatives. In this case the Jacobian is computed analytically via the user-supplied function.

NonlinearRegression does not actually store the Jacobian but uses fast Givens transformations to construct an orthogonal reduction of the Jacobian to upper triangular form. The reduction is based on fast Givens transformations (see Golub and Van Loan 1983, pages 156-162, Gentleman 1974). This method has two main advantages:

1. The loss of accuracy resulting from forming the crossproduct matrix used in the equations for $s_c$ is avoided.
2. The $n \times p$ Jacobian need not be stored saving space when $n > p$.

A weighted least squares fit can also be performed. This is appropriate when the variance of $\varepsilon_i$ in the nonlinear regression model is not constant but instead is $\sigma^2/w_i$. Here, $w_i$ are weights input via the user supplied function. For the weighted case, `NonlinearRegression` finds the estimate by minimizing a weighted sum of squares error.

### Programming Notes

Nonlinear regression allows users to specify the model’s functional form. This added flexibility can cause unexpected convergence problems for users who are unaware of the limitations of the algorithm. Also, in many cases, there are possible remedies that may not be immediately obvious. The following is a list of possible convergence problems and some remedies. No one-to-one correspondence exists between the problems and the remedies. Remedies for some problems may also be relevant for the other problems.

1. **A local minimum is found.** Try a different starting value. Good starting values can often be obtained by fitting simpler models. For example, for a nonlinear function
   \[ f(x; \theta) = \theta_1 e^{\theta_2 x} \]
   good starting values can be obtained from the estimated linear regression coefficients $\hat{\theta}_0$ and $\hat{\theta}_1$ from a simple linear regression of $\ln y$ on $\ln x$. The starting values for the nonlinear regression in this case would be
   \[ \theta_1 = e^{\hat{\theta}_0} \text{ and } \theta_2 = \hat{\theta}_1 \]
   If an approximate linear model is unclear, then simplify the model by reducing the number of nonlinear regression parameters. For example, some nonlinear parameters for which good starting values are known could be set to these values. This simplifies the approach to computing starting values for the remaining parameters.

2. **The estimate of $\theta$ is incorrectly returned as the same or very close to the initial estimate.**
   - The scale of the problem may be orders of magnitude smaller than the assumed default of 1 causing premature stopping. For example, if the sums of squares for error is less than approximately $(2.22e^{-16})^2$, the routine stops. See Example 3, which shows how to shut down some of the stopping criteria that may not be relevant for your particular problem and which also shows how to improve the speed of convergence by the input of the scale of the model parameters.
   - The scale of the problem may be orders of magnitude larger than the assumed default causing premature stopping. The information with regard to the input of the scale of the model parameters in Example 3 is also relevant here. In addition, the maximum allowable step size `Imsl.Stat.NonlinearRegression.MaxStepsize` (p. 654) in Example 3 may need to be increased.
   - The residuals are input with accuracy much less than machine accuracy, causing premature stopping because a local minimum is found. Again see Example 3 to see how to change some default tolerances. If you cannot improve the precision of the computations of the residual, you need to use method `Imsl.Stat.NonlinearRegression.Digits` (p. 652) to indicate the actual number of good digits in the residuals.
3. The model is discontinuous as a function of $\theta$. There may be a mistake in the user-supplied function. Note that the function $f(x; \theta)$ can be a discontinuous function of $x$.


5. Overflow occurs during the computations. Make sure the user-supplied functions do not overflow at some value of $\theta$.

6. The estimate of $\theta$ is going to infinity. A parameterization of the problem in terms of reciprocals may help.

7. Some components of $\theta$ are outside known bounds. This can sometimes be handled by making a function that produces artificially large residuals outside of the bounds (even though this introduces a discontinuity in the model function).

Note that the Imsl.Stat.NonlinearRegression.Solve(Imsl.Stat.NonlinearRegression.IFunction) (p. 657) method must be called before using any property as a right operand, otherwise the value is null.

**Properties**

**AbsoluteTolerance**

virtual public double AbsoluteTolerance {set; }

**Description**
The absolute function tolerance.

**Property Value**
A double scalar value specifying the absolute function tolerance.

**Remarks**
The tolerance must be greater than or equal to zero. By default, AbsoluteTolerance = 4.93e-32.

**Exception**
System.ArgumentException is thrown if AbsoluteTolerance is set less than 0

**Coefficients**

virtual public double[] Coefficients {get; }

**Description**
The regression coefficients.
Property Value
A double array containing the regression coefficients or null if
called.

DFError
virtual public double DFError {get; }

Description
The degrees of freedom for error.

Property Value
A double which specifies the degrees of freedom for error or null if
called.

Digits
virtual public int Digits {set; }

Description
The number of good digits in the residuals.

Property Value
An int specifying the number of good digits in the residuals.

Remarks
The number of digits must be greater than zero. By default, Digits = 15.

Exception
System.ArgumentException is thrown if Digits is set less than or equal to 0

ErrorStatus
virtual public int ErrorStatus {get; }

Description
Characterizes the performance of NonlinearRegression.

Property Value
An int specifying information about convergence.
### Remarks

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All convergence tests were met.</td>
</tr>
<tr>
<td>1</td>
<td>Scaled step tolerance was satisfied. The current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or StepTolerance is too big.</td>
</tr>
<tr>
<td>2</td>
<td>Scaled actual and predicted reductions in the function are less than or equal to the relative function convergence tolerance RelativeTolerance.</td>
</tr>
<tr>
<td>3</td>
<td>Iterates appear to be converging to a noncritical point. Incorrect gradient information, a discontinuous function, or stopping tolerances being too tight may be the cause.</td>
</tr>
<tr>
<td>4</td>
<td>Five consecutive steps with the maximum stepsize have been taken. Either the function is unbounded below, or has a finite asymptote in some direction, or the MaxStepsize is too small.</td>
</tr>
</tbody>
</table>

### See Also:
RelativeTolerance (p. 655), StepTolerance (p. 656), MaxStepsize (p. 654)

### FalseConvergenceTolerance

#### virtual public double FalseConvergenceTolerance {set; }

**Description**
The false convergence tolerance.

**Property Value**
A double scalar value specifying the false convergence tolerance.

**Remarks**
The tolerance must be greater than or equal to zero. By default, FalseConvergenceTolerance = 2.22e-14.

**Exception**
System.ArgumentException is thrown if FalseConvergenceTolerance is set less than 0.

### GradientTolerance

#### virtual public double GradientTolerance {set; }

**Description**
The gradient tolerance.

**Property Value**
A double specifying the gradient tolerance used to compute the gradient.

**Remarks**
The tolerance must be greater than or equal to zero. By default, GradientTolerance = 6.055e-6.

**Exception**
System.ArgumentException is thrown if GradientTolerance is set less than 0.

### Guess

#### virtual public double[] Guess {set; }

Regression NonlinearRegression
Description
The initial guess of the parameter values.

Property Value
A double array of initial values for the parameters.

Remarks
By default, Guess is an array of zeroes.

InitialTrustRegion
virtual public double InitialTrustRegion {set; }

Description
The initial trust region radius.

Property Value
A double scalar value specifying the initial trust region radius.

Remarks
The initial trust radius must be greater than zero. By default, InitialTrustRegion is set based on the initial scaled Cauchy step.

Exception
System.ArgumentException is thrown if InitialTrustRegion is less than or equal to 0

MaxIterations
virtual public int MaxIterations {set; }

Description
The maximum number of iterations allowed during optimization.

Property Value
An int specifying the maximum number of iterations allowed during optimization.

Remarks
The value must be greater than 0. By default, MaxIterations = 100.

Exception
System.ArgumentException is thrown if MaxIterations is set less than or equal to 0

MaxStepsize
virtual public double MaxStepsize {set; }

Description
The maximum allowable stepsize.

Property Value
A nonnegative double value specifying the maximum allowable stepsize.
Remarks
The maximum allowable stepsize must be greater than zero.
If this property is not set then the maximum stepsize is set to a default value based on a scaled $\theta$.

Exception
System.ArgumentException is thrown if MaxStepsize is less than or equal to 0

$R$
virtual public double[,] R {get; }

Description
A copy of the $R$ matrix.

Property Value
A two dimensional double array containing a copy of the $R$ matrix or null if

Remarks
The upper triangular matrix containing the $R$ matrix from a QR decomposition of the matrix of regressors.

$\text{Rank}$
virtual public int Rank {get; }

Description
The rank of the matrix.

Property Value
An int which specifies the rank of the matrix or null if

RelativeTolerance
virtual public double RelativeTolerance {set; }

Description
The relative function tolerance

Property Value
A double scalar value specifying the relative function tolerance.

Remarks
The relative function tolerance must be greater than or equal to zero. By default, RelativeTolerance = 1.0e-20.
Exception

  System.ArgumentException is thrown if RelativeTolerance is set less than 0

Scale
virtual public double[] Scale {set; }

Description
The scaling array for theta.

Property Value
A double array containing the scaling values for the parameters (theta).

Remarks
The elements of the scaling array must be greater than zero. Scale is used mainly in scaling the gradient and the distance between points. If good starting values of theta are known and are nonzero, then a good choice is Scale[i]=1.0/theta[i]. Otherwise, if theta is known to be in the interval (-10.e5, 10.e5), set Scale[i]=10.e-5. By default, the elements of the scaling array are set to 1.0.

By default, Scale is an array of ones.

Exception

  System.ArgumentException is thrown if any of the elements of Scale is set less than or equal to 0

StepTolerance
virtual public double StepTolerance {set; }

Description
The step tolerance.

Property Value
A double scalar value specifying the step tolerance used to step between two points.

Remarks
The step tolerance must be greater than or equal to zero. By default StepTolerance = 3.667e-11.

Exception

  System.ArgumentException is thrown if StepTolerance is set less than 0

Constructor

NonlinearRegression
public NonlinearRegression(int nparm)

Description
Constructs a new nonlinear regression object.
Parameter

\texttt{nparm} – An \texttt{int} which specifies the number of unknown parameters in the regression.

Methods

\textbf{GetCoefficient}\\
\texttt{virtual public double GetCoefficient(int i)}

\textbf{Description}\\
Returns the estimate for a coefficient.

\textbf{Parameter}\\
\texttt{i} – An \texttt{int} which specifies the index of a coefficient whose estimate is to be returned.

\textbf{Returns}\\
A \texttt{double} which contains the estimate for the \texttt{i}-th coefficient or \texttt{null} if \texttt{Imsl.Stat.NonlinearRegression.Solve(Imsl.Stat.NonlinearRegression.IFunction)} (p. 657) has not been called.

\textbf{GetSSE}\\
\texttt{virtual public double GetSSE()}

\textbf{Description}\\
Returns the sums of squares for error.

\textbf{Returns}\\
A \texttt{double} which contains the sum of squares for error or \texttt{null} if \texttt{Imsl.Stat.NonlinearRegression.Solve(Imsl.Stat.NonlinearRegression.IFunction)} (p. 657) has not been called.

\textbf{Solve}\\
\texttt{virtual public double[] Solve(Imsl.Stat.NonlinearRegression.IFunction \textit{F})}

\textbf{Description}\\
Solves the least squares problem and returns the regression coefficients.

\textbf{Parameter}\\
\texttt{\textit{F}} – An \texttt{Imsl.Stat.NonlinearRegression.IFunction} (p. 663) whose coefficients are to be computed.

\textbf{Returns}\\
A \texttt{double} array containing the regression coefficients.

\textbf{Exceptions}\\
\texttt{Imsl.Stat.TooManyIterationsException} is thrown when the number of allowed iterations is exceeded\n\texttt{Imsl.Stat.NegativeFreqException} is thrown when the specified frequency is negative\n\texttt{Imsl.Stat.NegativeWeightException} is thrown when the weight is negative
Example 1: Nonlinear Regression using Finite Differences

In this example a nonlinear model is fitted. The derivatives are obtained by finite differences.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;
public class NonlinearRegressionEx1 : NonlinearRegression.IFunction
{
    public bool f(double[] theta, int iobs, double[] frq, double[] wt, double[] e)
    {
        double[] ydata = new double[]{ 54.0, 50.0, 45.0, 37.0, 35.0,
                                        25.0, 20.0, 16.0, 18.0, 13.0,
                                        8.0, 11.0, 8.0, 4.0, 6.0};
        double[] xdata = new double[]{ 2.0, 5.0, 7.0, 10.0, 14.0,
                                        19.0, 26.0, 31.0, 34.0, 38.0,
                                        45.0, 52.0, 53.0, 60.0, 65.0};

        bool iend;
        int nobs = 15;
        if (iobs < nobs)
        {
            wt[0] = 1.0;
            frq[0] = 1.0;
            iend = true;
        }
        else
        {
            iend = false;
        }
        return iend;
    }
    public static void Main(String[] args)
    {
        int nparm = 2;
        double[] theta = new double[]{60.0, - 0.03};
        NonlinearRegression regression = new NonlinearRegression(nparm);
        regression.Guess = theta;
        NonlinearRegression.IFunction fcn = new NonlinearRegressionEx1();
        double[] coef = regression.Solve(fcn);

        Console.Out.WriteLine
        ("The computed regression coefficients are {" + coef[0] + ", ",
         + coef[1] + "}");
        Console.Out.WriteLine("The computed rank is " + regression.Rank);
        Console.Out.WriteLine("The degrees of freedom for error are " +
                          regression.DFError);
        Console.Out.WriteLine("The sums of squares for error is "
                       + regression.GetSSE());
        new PrintMatrix("R from the QR decomposition ").Print(regression.R);
    }
}
```
Output

The computed regression coefficients are \{58.6065629318535, -0.039586447289031\}
The computed rank is 2
The degrees of freedom for error are 13
The sums of squares for error is 49.459299862472

R from the QR decomposition

\[ \begin{array}{cc}
0 & 1 \\
0 & 1.87385998175942 & 1139.92836180938 \\
1 & 0 & 1139.79754530433 \\
\end{array} \]

Example 2: Nonlinear Regression with User-supplied Derivatives

In this example a nonlinear model is fitted. The derivatives are supplied by the user.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;
public class NonlinearRegressionEx2 : NonlinearRegression.IDerivative
{

double[] ydata = new double[]{
    54.0, 50.0, 45.0, 37.0, 35.0, 25.0, 20.0,
    16.0, 18.0, 13.0, 8.0, 11.0, 8.0, 4.0, 6.0};
double[] xdata = new double[]{
    2.0, 5.0, 7.0, 10.0, 14.0, 19.0, 26.0, 31.0,
    34.0, 38.0, 45.0, 52.0, 53.0, 60.0, 65.0};
bool iend;
int nobs = 15;

public bool f(double[] theta, int iobs, double[] frq, double[] wt, double[] e)
{
    if (iobs < nobs)
    {
        wt[0] = 1.0;
        frq[0] = 1.0;
        iend = true;
    }
    else
    {
        iend = false;
    }
    return iend;
}

public bool derivative(double[] theta, int iobs, double[] frq,
double[] wt, double[] de)
{
    if (iobs < nobs)
    {
        wt[0] = 1.0;
        frq[0] = 1.0;
        iend = true;
    } else
    {
        iend = false;
    }
    return iend;
}

public static void Main(String[] args)
{
    int nparm = 2;
    double[] theta = new double[]{60.0, - 0.03};
    NonlinearRegression regression = new NonlinearRegression(nparm);
    regression.Guess = theta;
    double[] coef = regression.Solve(new NonlinearRegressionEx2());
    Console.Out.WriteLine("The computed regression coefficients are 
{" + coef[0] + ", " + coef[1] + "}");
    Console.Out.WriteLine("The computed rank is "+ regression.Rank);
    Console.Out.WriteLine("The degrees of freedom for error are "+ regression.DFError);
    Console.Out.WriteLine("The sums of squares for error is "+ regression.GetSSE());
    new PrintMatrix("R from the QR decomposition ").Print(regression.R);
}

Output
The computed regression coefficients are {58.6065629254192, -0.0395864472775247}
The computed rank is 2
The degrees of freedom for error are 13
The sums of squares for error is 49.4592998624722
R from the QR decomposition
0 1
0 1.87385998422826 1139.79757620697
1 0 1139.79757620697

Example 3: NonlinearRegression using Set Methods
In this example, some nondefault tolerances and scales are used to fit a nonlinear model. The data is
1.e-10 times the data of Example 1. In order to fit this model without rescaling the data, we first set the
absolute function tolerance to 0.0. The default value would cause the program to terminate after one iteration because the residual sum of squares is roughly 1.e-19. We also set the relative function tolerance to 0.0. The gradient tolerance is properly scaled for this problem so we leave it at its default value. Finally, we set the elements of scale to the absolute value of the reciprocal of the starting value. The derivatives are obtained by finite differences.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;
public class NonlinearRegressionEx3 : NonlinearRegression.IFunction
{
    public bool f(double[] theta, int iobs, double[] frq, double[] wt, double[] e)
    {
        double[] ydata = new double[]{
            54e-10, 50e-10, 45e-10, 37e-10, 35e-10, 25e-10, 20e-10,
            16e-10, 18e-10, 13e-10, 8e-10, 11e-10, 8e-10, 4e-10, 6e-10};
        double[] xdata = new double[]{
            2.0, 5.0, 7.0, 10.0, 14.0, 19.0, 26.0, 31.0, 34.0, 38.0,
            45.0, 52.0, 53.0, 60.0, 65.0};
        bool iend;
        int nobs = 15;
        if (iobs < nobs)
        {
            wt[0] = 1.0;
            frq[0] = 1.0;
            iend = true;
        }
        else
        {
            iend = false;
        }
        return iend;
    }

    public static void Main(String[] args)
    {
        int nparm = 2;
        double[] theta = new double[]{6e-9, - 0.03};
        double[] scale = new double[nparm];
        NonlinearRegression regression = new NonlinearRegression(nparm);
        regression.Guess = theta;
        regression.AbsoluteTolerance = 0.0;
        regression.RelativeTolerance = 0.0;
        scale[0] = 1.0 / Math.Abs(theta[0]);
        scale[1] = 1.0 / Math.Abs(theta[1]);
        regression.Scale = scale;
        NonlinearRegression.IFunction fcn = new NonlinearRegressionEx3();
        double[] coef = regression.Solve(fcn);
        Console.Out.WriteLine("The computed regression coefficients are \{" +
            coef[0] + ", " + coef[1] + "}");
        Console.Out.WriteLine("The computed rank is " + regression.Rank);
        Console.Out.WriteLine("The degrees of freedom for error are " +
            regression.DFError);
        Console.Out.WriteLine("The sums of squares for error is " +
            regression.GetSSE());
    }
}```
new PrintMatrix("R from the QR decomposition ").Print(regression.R);
}
}

Output
The computed regression coefficients are \{5.78378362108798E-09, -0.0396252538296399\}
The computed rank is 2
The degrees of freedom for error are 13
The sums of squares for error is 5.16637661043416E-19

\[
\begin{array}{cc}
0 & 1.87310563212442 \\
1 & 0.58371399105394E-11
\end{array}
\]

Imsl.Stat.NonlinearRegression: Scaled step tolerance was satisfied. The current point may be an approximate local solution.

NonlinearRegression.IDerivative Interface

public interface Imsl.Stat.NonlinearRegression.IDerivative :
Imsl.Stat.NonlinearRegression.IFunction

Public interface for the user supplied function to compute the derivative for NonlinearRegression.

Method

derivative

abstract public bool derivative(double[] theta, int iobs, double[] frq,
double[] wt, double[] de)

Description
Computes the weight, frequency, and partial derivatives of the residual given the parameter vector \( \theta \) for a single observation.

Parameters
- \( \theta \) – An input double array which contains the parameter values of the regression function.
- \( \text{iobs} \) – An input int value indicating the observation index.
- \( \text{frq} \) – An output double array of length 1 containing the frequency for observation \( y[\text{iobs}] \).
- \( \text{wt} \) – An output double array of length 1 containing the weight for the observation \( y[\text{iobs}] \).
- \( \text{de} \) – An output double array containing the partial derivatives of the error (residual) for observation \( y[\text{iobs}] \).
Returns
A bool value representing the completion indicator. true indicates iobs is less than the number of observations. false indicates iobs is greater than or equal to the number of observations and wt, freq, and de are not output.

Remarks
The length of theta corresponds to the number of unknown parameters in the regression function. The function is evaluated at observation y[iobs]. Use wt = 1.0 for equal weighting (unweighted least squares). The length of de corresponds to the number of unknown parameters in the regression function.

NonlinearRegression.IFunction Interface

public interface Imsl.Stat.NonlinearRegression.IFunction
Public interface for the user supplied function for NonlinearRegression.

Method

f
abstract public bool f(double[] theta, int iobs, double[] frq, double[] wt, double[] e)

Description
Computes the weight, frequency, and residual given the parameter vector theta for a single observation.

Parameters

theta – An input double array containing the parameter values of the model.
iobs – An input int value indicating the observation index.
frq – An output double array of length 1 containing the frequency for observation y[iobs].
wt – An output double array of length 1 containing the weight for observation y[iobs].
e – An output double array of length 1 which contains the error (residual) for observation y[iobs].

Returns
A bool value representing the completion indicator. true indicates iobs is less than the number of observations. false indicates iobs is greater than or equal to the number of observations and wt, freq, and e are not output.
Remarks
The length of theta corresponds to the number of unknown parameters in the model.
The function is evaluated at observation y[iobs].
Use wt = 1.0 for equal weighting (unweighted least squares).

SelectionRegression Class

public class Imsl.Stat.SelectionRegression

Selects the best multiple linear regression models.

Class SelectionRegression finds the best subset regressions for a regression problem with three or more independent variables. Typically, the intercept is forced into all models and is not a candidate variable. In this case, a sum-of-squares and crossproducts matrix for the independent and dependent variables corrected for the mean is computed internally. Optionally, SelectionRegression supports user-calculated sum-of-squares and crossproducts matrices; see the description of the Imsl.Stat.SelectionRegression.Compute(System.Double[,],System.Double[]) (p. 668) method.

“Best” is defined by using one of the following three criteria:

- $R^2$ (in percent)
  \[ R^2 = 100 \left(1 - \frac{SSE_p}{SST}\right) \]

- $R^2_a$ (adjusted $R^2$)
  \[ R^2_a = 100\left[1 - \frac{n - 1}{n - p} \frac{SSE_p}{SST}\right] \]

  Note that maximizing the $R^2_a$ is equivalent to minimizing the residual mean squared error:
  \[ \frac{SSE_p}{(n - p)} \]

- Mallow’s $C_p$ statistic
  \[ C_p = \frac{SSE_p}{\hat{\sigma}_k^2} + 2p - n \]

Here, $n$ is equal to the sum of the frequencies (or the number of rows in x if frequencies are not specified in the Compute method), and SST is the total sum-of-squares. $k$ is the number of candidate or independent variables, represented as the nCandidate argument in the SelectionRegression constructor. $SSE_p$ is the error sum-of-squares in a model containing $p$ regression parameters including $\beta_0$ (or $p - 1$ of the $k$ candidate variables). Variable $\hat{\sigma}_k^2$
is the error mean square from the model with all $k$ variables in the model. Hocking (1972) and Draper and Smith (1981, pp. 296-302) discuss these criteria.

Class SelectionRegression is based on the algorithm of Furnival and Wilson (1974). This algorithm finds the maximum number of good saved candidate regressions for each possible subset size. For more details, see method MaximumGoodSaved. These regressions are used to identify a set of best regressions. In large problems, many regressions are not computed. They may be rejected without computation based on results for other subsets; this yields an efficient technique for considering all possible regressions.

There are cases when the user may want to input the variance-covariance matrix rather than allow it to be calculated. This can be accomplished using the appropriate Compute method. Three situations in which the user may want to do this are as follows:

1. The intercept is not in the model. A raw (uncorrected) sum of squares and crossproducts matrix for the independent and dependent variables is required. Argument numberOfObservations must be set to 1 greater than the number of observations. Form $A^T A$, where $A = [A, Y]$, to compute the raw sum-of-squares and crossproducts matrix.

2. An intercept is a candidate variable. A raw (uncorrected) sum of squares and crossproducts matrix for the constant regressor (= 1.0), independent, and dependent variables is required for $\mathbf{cov}$. In this case, $\mathbf{cov}$ contains one additional row and column corresponding to the constant regressor. This row and column contain the sum-of-squares and crossproducts of the constant regressor with the independent and dependent variables. The remaining elements in $\mathbf{cov}$ are the same as in the previous case. Argument numberOfObservations must be set to 1 greater than the number of observations.

3. There are $m$ variables that must be forced into the models. A sum-of-squares and crossproducts matrix adjusted for the $m$ variables is required (calculated by regressing the candidate variables on the variables to be forced into the model). Argument numberOfObservations must be set to $m$ less than the number of observations.

**Programming Notes**

SelectionRegression can save considerable CPU time over explicitly computing all possible regressions. However, the function has some limitations that can cause unexpected results for users who are unaware of the limitations of the software.

1. For $k + 1 > - \log_2(\epsilon)$, where $\epsilon$ is the largest relative spacing for double precision, some results can be incorrect. This limitation arises because the possible models indicated (the model numbers 1, 2, ..., $2k$) are stored as floating-point values; for sufficiently large $k$, the model numbers cannot be stored exactly. On many computers, this means SelectionRegression (for $k > 49$) can produce incorrect results.

2. SelectionRegression eliminates some subsets of candidate variables by obtaining lower bounds on the error sum-of-squares from fitting larger models. First, the full model containing all independent variables is fit sequentially using a forward stepwise procedure in which one variable enters the model at a time, and criterion values and model numbers for all the candidate variables that can enter at each step are stored. If linearly dependent variables are removed from the full
model, a “VariablesDeleted” warning is issued. In this case, some submodels that contain variables removed from the full model because of linear dependency can be overlooked if they have not already been identified during the initial forward stepwise procedure. If this warning is issued and you want the variables that were removed from the full model to be considered in smaller models, you can rerun the program with a set of linearly independent variables.

Properties

**CriterionOption**

virtual public Imsl.Stat.SelectionRegression.Criterion CriterionOption {get; set; }

**Description**

The criterion option used to calculate the regression estimates.

**Property Value**

An int containing the criterion option.

**Remarks**

By default for all criteria, subset size 1, 2, ..., \( k = n_{\text{Candidate}} \) are considered. However, for \( R^2 \) the maximum number of subsets can be restricted using property Imsl.Stat.SelectionRegression.MaximumSubsetSize (p. 667).

<table>
<thead>
<tr>
<th>Criterion Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSquared</td>
<td>For ( R^2 ), subset sizes 1, 2, ..., ( n_{\text{Candidate}} ) are examined. This is the default with ( \text{MaximumSubsetSize} = n_{\text{Candidate}} ).</td>
</tr>
<tr>
<td>AdjustedRSquared</td>
<td>For Adjusted ( R^2 ), subset sizes 1, 2, ..., ( n_{\text{Candidate}} ) are examined.</td>
</tr>
<tr>
<td>MallowsCP</td>
<td>For Mallow’s ( C_p ) Subset sizes 1, 2, ..., ( n_{\text{Candidate}} ) are examined.</td>
</tr>
</tbody>
</table>

See Also: RSquared (p. 675), AdjustedRSquared (p. 675), MallowsCP (p. 675)

**MaximumBestFound**

virtual public int MaximumBestFound {set; }

**Description**

The maximum number of best regressions to be found.

**Property Value**

An int containing the maximum number of best regressions to be reported.
Remarks
If the $R^2$ criterion option is selected, the MaximumBestFound best regressions for each subset size examined are reported. If the adjusted $R^2$ or Mallow’s $C_p$ criteria are selected, the MaximumBestFound among all possible regressions are found.

By default, MaximumBestFound = 1.

See Also: RSquared (p. 675), AdjustedRSquared (p. 675), MallowsCP (p. 675)

**MaximumGoodSaved**
virtual public int MaximumGoodSaved {set; }

Description
The maximum number of good regressions for each subset size saved.

Property Value
An int containing the maximum number of good regressions saved for each subset size.

Remarks
MaximumGoodSaved must be greater than or equal to Imsl.Stat.SelectionRegression.MaximumBestFound (p. 666). Normally, MaximumGoodSaved should be less than or equal to 10. It should never need be larger than MaximumSubsetSize, the maximum number of subsets for any subset size. Computing time required is inversely related to MaximumGoodSaved.

The default value is maximum(10, Imsl.Stat.SelectionRegression.MaximumSubsetSize (p. 667)).

**MaximumSubsetSize**
virtual public int MaximumSubsetSize {set; }

Description
The maximum subset size if $R^2$ criterion is used.

Property Value
An int containing the maximum subset size when $R^2$ criterion is used.

Remarks
By default, MaximumSubsetSize = nCandidate.

See Also: RSquared (p. 675), AdjustedRSquared (p. 675), MallowsCP (p. 675)

**Statistics**
virtual public Imsl.Stat.SelectionRegression.SummaryStatistics Statistics {get; }

Description
A SummaryStatistics object.

Property Value
A SummaryStatistics object containing the Coefficient statistics.
Constructor

**SelectionRegression**

`public SelectionRegression(int nCandidate)`

**Description**

Constructs a new `SelectionRegression` object.

**Parameter**

- `nCandidate` – An `int` containing the number of candidate variables (independent variables).

**Remarks**

`nCandidate` must be greater than 2.

Methods

**Compute**

`virtual public void Compute(double[,] x, double[] y)`

**Description**

Computes the best multiple linear regression models.

**Parameters**

- `x` – A `double` matrix containing the observations of the candidate (independent) variables.
- `y` – A `double` array containing the observations of the dependent variable.

**Remarks**

The number of columns in `x` must be equal to the number of variables set in the constructor.

**Exceptions**

- `Imsl.Stat.NoVariablesException` is thrown if no variables can enter any model
- `Imsl.Stat.TooManyObsDeletedException` is thrown if more observations have been deleted than were originally entered
- `Imsl.Stat.MoreObsDelThanEnteredException` is thrown if more observations are being deleted from the output covariance matrix than were originally entered
- `Imsl.Stat.DiffObsDeletedException` is thrown if different observations are being deleted from the return matrix than were originally entered

**Compute**

`virtual public void Compute(double[,] x, double[] y, double[] weights)`

**Description**

Computes the best weighted multiple linear regression models.
Parameters

- `x` – A double matrix containing the observations of the candidate (independent) variables.
- `y` – A double array containing the observations of the dependent variable.
- `weights` – A double array containing the weight for each of the observations.

Remarks

The number of columns in `x` must be equal to the number of variables set in the constructor.

Exceptions

- `Imsl.Stat.NoVariablesException` is thrown if no variables can enter any model
- `Imsl.Stat.NegativeWeightException` is thrown if a weight is less than zero.
- `Imsl.Stat.TooManyObsDeletedException` is thrown if more observations have been deleted than were originally entered
- `Imsl.Stat.MoreObsDelThanEnteredException` is thrown if more observations are being deleted from the output covariance matrix than were originally entered
- `Imsl.Stat.DiffObsDeletedException` is thrown if different observations are being deleted from the return matrix than were originally entered

Compute

```java
virtual public void Compute(double[,] x, double[] y, double[] weights, double[] frequencies)
```

Description

Computes the best weighted multiple linear regression models using frequencies for each observation.

Parameters

- `x` – A double matrix containing the observations of the candidate (independent) variables.
- `y` – A double array containing the observations of the dependent variable.
- `weights` – A double array containing the weight for each of the observations.
- `frequencies` – A double array containing the frequency for each of the observations of `x`.

Remarks

The number of columns in `x` must be equal to the number of variables set in the constructor.

Exceptions

- `Imsl.Stat.NoVariablesException` is thrown if no variables can enter any model
- `Imsl.Stat.NegativeFreqException` is thrown if a frequency is less than zero.
- `Imsl.Stat.NegativeWeightException` is thrown if a weight is less than zero.
- `Imsl.Stat.TooManyObsDeletedException` is thrown if more observations have been deleted than were originally entered
- `Imsl.Stat.MoreObsDelThanEnteredException` is thrown if more observations are being deleted from the output covariance matrix than were originally entered
- `Imsl.Stat.DiffObsDeletedException` is thrown if different observations are being deleted from the return matrix than were originally entered
Imsl.Stat.DiffObsDeletedException is thrown if different observations are being deleted from the return matrix than were originally entered.

**Compute**

```csharp
virtual public void Compute(double[,] cov, int nObservations)
```

**Description**

Computes the best multiple linear regression models using a user-supplied covariance matrix.

**Parameters**

- **cov** – A `double` matrix containing a variance-covariance or sum-of-squares and crossproducts matrix, in which the last column must correspond to the dependent variable.
- **nObservations** – An `int` containing the number of observations used to compute `cov`.

**Remarks**

cov can be computed using the Imsl.Stat.Covariances (p. 545) class.

**Exception**

Imsl.Stat.NoVariablesException is thrown if no variables can enter any model.

**Example 1: SelectionRegression**

This example uses a data set from Draper and Smith (1981, pp. 629-630). Class `SelectionRegression` is invoked to find the best regression for each subset size using the $R^2$ criterion.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;

public class SelectionRegressionEx1
{
    public static void Main(String[] args)
    {
        double[,] x = {{7.0, 26.0, 6.0, 60.0},
                       {1.0, 29.0, 15.0, 52.0},
                       {11.0, 56.0, 8.0, 20.0},
                       {11.0, 31.0, 8.0, 47.0},
                       {7.0, 52.0, 6.0, 33.0},
                       {11.0, 55.0, 9.0, 22.0},
                       {3.0, 71.0, 17.0, 6.0},
                       {1.0, 31.0, 22.0, 44.0},
                       {2.0, 54.0, 18.0, 22.0},
                       {21.0, 47.0, 4.0, 26},
                       {1.0, 40.0, 23.0, 34.0},
                       {11.0, 66.0, 9.0, 12.0},
                       {10.0, 68.0, 8.0, 12.0}};
        double[] y = new double[] {78.5, 74.3, 104.3,}
    }
}
SelectionRegression sr = new SelectionRegression(4);
sr.Compute(x, y);

for (int i = 1; i <= 4; i++)
{
    double[] tmpCrit = stats.GetCriterionValues(i);
    int[,] indvar = stats.GetIndependentVariables(i);
    Console.WriteLine("Regressions with " + i + " variable(s) (R-squared)");
    for (int j = 0; j < tmpCrit.GetLength(0); j++)
    {
        Console.Write(" + tmpCrit[j] + " );
        for (int k = 0; k < indvar.GetLength(1); k++)
            Console.Write(indvar[j,k] + " ");
        Console.WriteLine("");
    }
    Console.WriteLine("");
}

// Setup a PrintMatrix object for use in the loop below.
PrintMatrix pm = new PrintMatrix();
pm.SetColumnSpacing(8);
PrintMatrixFormat tst = new PrintMatrixFormat();
tst.SetNoColumnLabels();
tst.SetNoRowLabels();
tst.NumberFormat = "0.000";
for (int i = 0; i < 4; i++)
{
    double[,] tmpCoef = stats.GetCoefficientStatistics(i);
    Console.WriteLine("Regressions with " + (i+1) + " variable(s) (R-squared)";
    Console.WriteLine("Variable Coefficient Standard Error t-statistic p-value" +
            "(R-squared)";
    Console.WriteLine("Variable Coefficient Standard Error t-statistic p-value" +
            "(R-squared)";
    pm.Print(tst, tmpCoef);
}

Output

Regressions with 1 variable(s) (R-squared)
67.454964131609 4
66.6288257633294 2
53.3948023835033 1
28.5872731229812 3

Regressions with 2 variable(s) (R-squared)
97.8678374535631 1 2
97.2471047716931 1 4
93.5289640615807 3 4
68.006040795005 2 4

Regression SelectionRegression ● 671
Regressions with 3 variable(s) (R-squared)
98.229541200426 1 2 4
98.1281092587343 1 3 4
97.2819959386273 2 3 4

Regressions with 4 variable(s) (R-squared)
98.237562040768 1 2 3 4

Regressions with 1 variable(s) (R-squared)
Variable Coefficient Standard Error t-statistic p-value
4.000 -0.738 0.155 -4.775 0.001

Regressions with 2 variable(s) (R-squared)
Variable Coefficient Standard Error t-statistic p-value
1.000 1.468 0.121 12.105 0.000
2.000 0.662 0.046 14.442 0.000

Regressions with 3 variable(s) (R-squared)
Variable Coefficient Standard Error t-statistic p-value
1.000 1.452 0.117 12.410 0.000
2.000 0.416 0.186 2.242 0.052
4.000 -0.237 0.173 -1.365 0.205

Regressions with 4 variable(s) (R-squared)
Variable Coefficient Standard Error t-statistic p-value
1.000 1.551 0.745 2.083 0.071
2.000 0.510 0.724 0.705 0.501
3.000 0.102 0.755 0.135 0.896
4.000 -0.144 0.709 -0.203 0.844

Example 2: SelectionRegression

This example uses the same data set as the first example, but Mallow’s $C_p$ statistic is used as the criterion rather than $R^2$. Note that when Mallow’s $C_p$ statistic (or adjusted $R^2$) is specified, MaximumBestFound is used to indicate the total number of “best” regressions (rather than indicating the number of best regressions per subset size, as in the case of the $R^2$ criterion). In this example, the three best regressions are found to be (1, 2), (1, 2, 4), and (1, 2, 3).

```csharp
using System;
```
using Imsl.Math;
using Imsl.Stat;

class SelectionRegressionEx2
{
    public static void Main(String[] args)
    {
        double[,] x = {
            {7.0, 26.0, 6.0, 60.0},
            {1.0, 29.0, 15.0, 52.0},
            {11.0, 56.0, 8.0, 20.0},
            {11.0, 31.0, 8.0, 47.0},
            {7.0, 52.0, 6.0, 33.0},
            {11.0, 55.0, 9.0, 22.0},
            {3.0, 71.0, 17.0, 6.0},
            {1.0, 31.0, 22.0, 44.0},
            {2.0, 54.0, 18.0, 22.0},
            {21.0, 47.0, 4.0, 26},
            {1.0, 40.0, 23.0, 34.0},
            {11.0, 66.0, 9.0, 12.0},
            {10.0, 68.0, 8.0, 12.0}
        };
        double[] y = new double[] { 78.5, 74.3, 104.3, 87.6,
            95.9, 109.2, 102.7, 72.5,
            93.1, 115.9, 83.8, 113.3,
            109.4};

        SelectionRegression sr = new SelectionRegression(4);
        sr.MaximumBestFound = 3;
        sr.Computed(x, y);
        SelectionRegression.SummaryStatistics stats = sr.Stats;

        for (int i = 1; i <= 4; i++)
        {
            double[] tmpCrit = stats.GetCriterionValues(i);
            int[,] indvar = stats.GetIndependentVariables(i);
            Console.WriteLine("Regressions with "+i+" variable(s) (MallowsCP)");
            for (int j = 0; j < tmpCrit.GetLength(0); j++)
            {
                Console.Write(" " + tmpCrit[j] + " ");
                for (int k = 0; k < indvar.GetLength(1); k++)
                    Console.Write(indvar[j,k] + " ");
                Console.WriteLine("");
            }
            Console.WriteLine("");
        }

        // Setup a PrintMatrix object for use in the loop below.
        PrintMatrix pm = new PrintMatrix();
        pm.SetColumnSpacing(9);
        PrintMatrixFormat tst = new PrintMatrixFormat();
        tst.SetNoColumnLabels();
        tst.SetNoRowLabels();
        tst.NumberFormat = "0.000";
        for (int i = 0; i < 3; i++)
        {
double[,] tmpCoeff = stats.GetCoefficientStatistics(i);
Console.Out.WriteLine("Regressions with "+(i+1)+" variable(s) (MallowsCP)\n\nVariable Coefficient Standard Error t-statistic p-value\n\npm.Print(tst, tmpCoeff);\n\nOutput\n\nRegressions with 1 variable(s) (MallowsCP)\n138.730833491674 4
142.486406936958 2
202.548769123445 1
315.154284140073 3
\nRegressions with 2 variable(s) (MallowsCP)\n2.6782415983184 1 2
5.4958508247584 1 4
22.3731119646967 3 4
138.225919754638 2 4
198.094652569569 1 3
\nRegressions with 3 variable(s) (MallowsCP)\n3.01823347348731 1 2 4
3.04127972306423 1 2 3
3.4963244234832 1 3 4
7.33747399565576 2 3 4
\nRegressions with 4 variable(s) (MallowsCP)\n5 1 2 3 4
\nRegressions with 1 variable(s) (MallowsCP)\nVariable Coefficient Standard Error t-statistic p-value\n1.000 1.468 0.121 12.105 0.000
2.000 0.662 0.046 14.442 0.000
\nRegressions with 2 variable(s) (MallowsCP)\nVariable Coefficient Standard Error t-statistic p-value\n1.000 1.452 0.117 12.410 0.000
2.000 0.416 0.186 2.242 0.052
4.000 -0.237 0.173 -1.365 0.205
\nRegressions with 3 variable(s) (MallowsCP)\nVariable Coefficient Standard Error t-statistic p-value
SelectionRegression.Criterion Enumeration

public enumeration Imsl.Stat.SelectionRegression.Criterion

Criterion Methods.

Fields

**AdjustedRSquared**


**Description**

Indicates $R^2_a$ (adjusted $R^2$) criterion regression.

**MallowsCP**

public Imsl.Stat.SelectionRegression.Criterion MallowsCP

**Description**

Indicates Mallow’s $C_p$ criterion regression.

**RSquared**

public Imsl.Stat.SelectionRegression.Criterion RSquared

**Description**

Indicates $R^2$ criterion regression.

SelectionRegression.SummaryStatistics Class

public class Imsl.Stat.SelectionRegression.SummaryStatistics

SummaryStatistics contains statistics related to the regression coefficients.
Methods

GetCoefficientStatistics
virtual public double[,] GetCoefficientStatistics(int regressionIndex)

Description
Returns the coefficients statistics for each of the best regressions found for each subset considered.

Parameter
regressionIndex – An int which specifies the index of the best regression statistics to return.

Returns
A two-dimensional double array containing the regression statistics.

Remarks
The value set using Imsl.Stat.SelectionRegression.MaximumBestFound (p. 666) determines the total number of best regressions to find. The number of best regression is equal to (Imsl.Stat.SelectionRegression.MaximumSubsetSize (p. 667) x MaximumBestFound), if criterion RSquared is specified or it is equal to MaximumBestFound if either MallowsCP or AdjustedRSquared is specified.

Each row contains statistics related to the regression coefficients of the best models. The regressions are ordered so that the better regressions appear first. The statistic in the columns are as follows (inferences are conditional on the selected model):

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>variable number</td>
</tr>
<tr>
<td>1</td>
<td>coefficient estimate</td>
</tr>
<tr>
<td>2</td>
<td>estimated standard error of the estimate</td>
</tr>
<tr>
<td>3</td>
<td>t-statistic for the test that the coefficient is 0</td>
</tr>
<tr>
<td>4</td>
<td>p-value for the two-sided t test</td>
</tr>
</tbody>
</table>

There will be 0 to (MaximumSubsetSize x MaximumBestFound - 1) best regressions if RSquared is specified or 0 to (MaximumBestFound - 1) if either MallowsCP or AdjustedRSquared is specified.

See Also: RSquared (p. 675), AdjustedRSquared (p. 675), MallowsCP (p. 675)

GetCriterionValues
virtual public double[] GetCriterionValues(int numVariables)

Description
Returns an array containing the values of the best criterion for the number of variables considered.

Parameter
numVariables – An int which specifies the number of variables considered.
Returns
A double array with \texttt{Imsl.Stat.SelectionRegression.MaximumSubsetSize} \ ((\textit{p. 667}) rows and \texttt{nCandidate} columns containing the criterion values.

\textbf{GetIndependentVariables}

\texttt{virtual public int[,] GetIndependentVariables(int numVariables)}

\textbf{Description}

Returns the identification numbers for the independent variables for the number of variables considered and in the same order as the criteria returned by \texttt{Imsl.Stat.SelectionRegression.SummaryStatistics.GetCriterionValues(System.Int32)} \ ((\textit{p. 676})).

\textbf{Parameter}

\texttt{numVariables} – An \texttt{int} which specifies the number of variables considered.

\textbf{Returns}

An \texttt{int} array containing the identification numbers for the independent variables considered.

\section*{StepwiseRegression Class}

\texttt{public class Imsl.Stat.StepwiseRegression}

Builds multiple linear regression models using forward selection, backward selection, or stepwise selection.

Class \texttt{StepwiseRegression} builds a multiple linear regression model using forward selection, backward selection, or forward stepwise (with a backward glance) selection.

Levels of priority can be assigned to the candidate independent variables using \texttt{Imsl.Stat.StepwiseRegression.Levels} \ ((\textit{p. 680}). All variables with a priority level of 1 must enter the model before variables with a priority level of 2. Similarly, variables with a level of 2 must enter before variables with a level of 3, etc. Variables also can be forced into the model using \texttt{Imsl.Stat.StepwiseRegression.Force} \ ((\textit{p. 679}). Note that specifying “force” without also specifying levels of priority will result in all variables being forced into the model.

Typically, the intercept is forced into all models and is not a candidate variable. In this case, a sum-of-squares and crossproducts matrix for the independent and dependent variables corrected for the mean is required. Other possibilities are as follows:

1. The intercept is not in the model. A raw (uncorrected) sum-of-squares and crossproducts matrix for the independent and dependent variables is required as input in \texttt{cov}. Argument \texttt{nObservations} must be set to one greater than the number of observations.

2. An intercept is a candidate variable. A raw (uncorrected) sum-of-squares and crossproducts matrix for the constant regressor (=1), independent and dependent variables are required for \texttt{cov}. In this...
case, \texttt{cov} contains one additional row and column corresponding to the constant regressor. This row/column contains the sum-of-squares and crossproducts of the constant regressor with the independent and dependent variables. The remaining elements in \texttt{cov} are the same as in the previous case. Argument \texttt{nObservations} must be set to one greater than the number of observations.

The stepwise regression algorithm is due to Efroymson (1960). \texttt{StepwiseRegression} uses sweeps of the covariance matrix (input in \texttt{cov}, if the covariance matrix is specified, or generated internally) to move variables in and out of the model (Hemmerle 1967, Chapter 3). The \texttt{SWEEP} operator discussed in Goodnight (1979) is used. A description of the stepwise algorithm is also given by Kennedy and Gentle (1980, pp. 335-340). The advantage of stepwise model building over all possible regression (\texttt{SelectionRegression}) is that it is less demanding computationally when the number of candidate independent variables is very large. However, there is no guarantee that the model selected will be the best model (highest $R^2$) for any subset size of independent variables.

**Properties**

**ANOVA**

virtual public Imsl.Stat.ANOVA ANOVA {get; }

**Description**

An analysis of variance table and related statistics.

**Property Value**

An \texttt{ANOVA} table and related statistics.

**CoefficientTTests**

virtual public Imsl.Stat.StepwiseRegression.CoefficientTTestsValue CoefficientTTests {get; }

**Description**

The student-$t$ test statistics for the regression coefficients.

**Property Value**

A \texttt{StepwiseRegression.CoefficientTTestsValue} object containing statistics relating to the regression coefficients.

**CoefficientVIF**

virtual public double[] CoefficientVIF {get; }

**Description**

The variance inflation factors for the final model in this invocation.

**Property Value**

A double array containing the variance inflation factors for the final model in this invocation.
Remarks
The elements are in the same order as the independent variables in \( x \) (or, if the covariance matrix is specified, the elements are in the same order as the variables in \( \text{cov} \)). Each element corresponding to a variable not in the model contains statistics for a model which includes the variables of the final model and the variables corresponding to the element in question.

The square of the multiple correlation coefficient for the \( i \)-th regressor after all others can be obtained from the \( i \)-th element for the returned array by the following formula:

\[
1.0 - \frac{1.0}{VIF}
\]

CovariancesSwept

\[ \text{virtual public double[,] CovariancesSwept \{get; \}} \]

Description
Results after \( \text{cov} \) has been swept for the columns corresponding to the variables in the model.

Property Value
A double matrix containing the results after \( \text{cov} \) has been swept on the columns corresponding to the variables in the model.

Remarks
The estimated variance-covariance matrix of the estimated regression coefficients in the final model can be obtained by extracting the rows and columns corresponding to the independent variables in the final model and multiplying the elements of this matrix by the error mean square.

Force

\[ \text{virtual public int Force \{set; \}} \]

Description
Forces independent variables into the model based on their level assigned from \( \text{Levels} \).

Property Value
An int specifying the upper bound on the variables forced into the model.

Remarks
Variables with levels 1, 2, ..., Force are forced into the model as independent variables.

See Also: Levels (p. 680)

History

\[ \text{virtual public double[] History \{get; \}} \]

Description
The stepwise regression history for the independent variables.

Property Value
A double array containing the recent history of the independent variables.
Remarks
The last element corresponds to the dependent variable.

<table>
<thead>
<tr>
<th>History[i]</th>
<th>Status of i-th Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>This variable has never been added to the model.</td>
</tr>
<tr>
<td>0.5</td>
<td>This variable was added into the model during initialization.</td>
</tr>
<tr>
<td>k &gt; 0.0</td>
<td>This variable was added to the model during the k-th step.</td>
</tr>
<tr>
<td>k &lt; 0.0</td>
<td>This variable was deleted from model during the k-th step</td>
</tr>
</tbody>
</table>

See Also: Levels (p. 680)

Intercept
virtual public double Intercept {get; }

Description
Returns the intercept.

Remarks
The intercept is computed as follows:

\[ \beta_0 = \bar{y} - \sum_{i=1}^{n} \beta_i \bar{x}_i - 1 \]

where \( \bar{y} \) is the mean of the dependent variable \( y \), \( \beta_i \) are the coefficients, and \( \bar{x}_i \) are the mean values for each independent variable \( x_i \) in the final model. If the covariance matrix is used for input, use method SetMean to specify the means of the variables. If \( x \) and \( y \) are used for input, the means are computed internally and do not need to be specified.

Levels
virtual public int[] Levels {set; }

Description
The levels of priority for variables entering and leaving the regression.

Property Value
An int array containing the levels of entry into the model for each variable.

Remarks
Each variable is assigned a positive value which indicates its level of entry into the model. A variable can enter the model only after all variables with smaller nonzero levels of entry have entered. Similarly, a variable can only leave the model after all variables with higher levels of entry have left. Variables with the same level of entry compete for entry (deletion) at each step. A value Levels[i]=0 means the i-th variable never enters the model. A value Levels[i]=-1 means the i-th variable is the dependent variable. The last element in Levels must correspond to the dependent variable, except when the variance-covariance or sum-of-squares and crossproducts matrix is supplied.

By default, Levels = {1, 1, ..., 1, -1}, where -1 corresponds to the dependent variable.

See Also: Force (p. 679)

Method
virtual public Imsl.Stat.StepwiseRegression.Direction Method {set; }

680 • StepwiseRegression

IMSL C# Numerical Library
**Description**
Specifies the stepwise selection method, forward, backward, or stepwise Regression.

**Property Value**
An int value between -1 and 1 specifying the stepwise selection method.

**Remarks**
Fields Forward, Backward, and Stepwise should be used.
By default, Direction.Stepwise.
See Also: Forward (p. 688), Backward (p. 688), Stepwise (p. 688)

---

**PValueIn**
virtual public double PValueIn {set; }

**Description**
Defines the largest $p$-value for variables entering the model.

**Property Value**
A double containing the largest $p$-value for variables entering the model.

**Remarks**
Variables with $p$-value less than PValueIn may enter the model. Backward regression does not use this value.
By default, PValueIn = 0.05.

---

**PValueOut**
virtual public double PValueOut {set; }

**Description**
Defines the smallest $p$-value for removing variables.

**Property Value**
A double containing the smallest $p$-value for removing variables from the model.

**Remarks**
Variables with $p$-values greater than PValueOut may leave the model. PValueOut must be greater than or equal to PValueIn. A common choice for PValueOut is $2 \times$PValueIn. Forward regression does not use this value.
By default, PValueOut = 0.10.

---

**Swept**
virtual public double[] Swept {get; }  

**Description**
An array containing information indicating whether or not a particular variable is in the model.

**Property Value**
A double array with information to indicate the independent variables in the model.
Remarks
The last element corresponds to the dependent variable. A +1 in the \( i \)-th position indicates that the variable is in the selected model. A -1 indicates that the variable is not in the selected model.

See Also: Levels (p. 680)

**Tolerance**
```
virtual public double Tolerance {set;}
```

**Description**
The tolerance used to detect linear dependence among the independent variables.

**Property Value**
A `double` containing the tolerance used for detecting linear dependence.

**Remarks**
By default, `Tolerance = 2.2204460492503e-16`.

---

**Constructors**

**StepwiseRegression**
```
public StepwiseRegression(double[,] x, double[] y)
```

**Description**
Creates a new instance of `StepwiseRegression`.

**Parameters**
- `x` – A double matrix of \( nObs \) by \( nVars \), where \( nObs \) is the number of observations and \( nVars \) is the number of independent variables.
- `y` – A double array containing the observations of the dependent variable.

**Exceptions**
- `Imsl.Stat.TooManyObsDeletedException` is thrown if more observations have been deleted than were originally entered
- `Imsl.Stat.MoreObsDelThanEnteredException` is thrown if more observations are being deleted from the output covariance matrix than were originally entered
- `Imsl.Stat.DiffObsDeletedException` is thrown if different observations are being deleted from the return matrix than were originally entered

**StepwiseRegression**
```
public StepwiseRegression(double[,] x, double[] y, double[] weights)
```

**Description**
Creates a new instance of weighted `StepwiseRegression`. 
Parameters

- **x** – A double matrix of \( n_{Obs} \) by \( n_{Vars} \), where \( n_{Obs} \) is the number of observations and \( n_{Vars} \) is the number of independent variables.
- **y** – A double array containing the observations of the dependent variable.
- **weights** – A double array containing the weight for each observation of \( x \).

Exceptions

- `Imsl.Stat.TooManyObsDeletedException` is thrown if more observations have been deleted than were originally entered.
- `Imsl.Stat.MoreObsDelThanEnteredException` is thrown if more observations are being deleted from the output covariance matrix than were originally entered.
- `Imsl.Stat.DiffObsDeletedException` is thrown if different observations are being deleted from the return matrix than were originally entered.
- `Imsl.Stat.NegativeWeightException` is thrown if a weight is less than zero.

StepwiseRegression

```
public StepwiseRegression(double[,] x, double[] y, double[] weights, double[] frequencies)
```

Description

Creates a new instance of weighted StepwiseRegression using observation frequencies.

Parameters

- **x** – A double matrix of \( n_{Obs} \) by \( n_{Vars} \), where \( n_{Obs} \) is the number of observations and \( n_{Vars} \) is the number of independent variables.
- **y** – A double array containing the observations of the dependent variable.
- **weights** – A double array containing the weight for each observation of \( x \).
- **frequencies** – A double array containing the frequency for each row of \( x \).

Exceptions

- `Imsl.Stat.TooManyObsDeletedException` is thrown if more observations have been deleted than were originally entered.
- `Imsl.Stat.MoreObsDelThanEnteredException` is thrown if more observations are being deleted from the output covariance matrix than were originally entered.
- `Imsl.Stat.DiffObsDeletedException` is thrown if different observations are being deleted from the return matrix than were originally entered.
- `Imsl.Stat.NegativeWeightException` is thrown if a weight is less than zero.
- `Imsl.Stat.NegativeFreqException` is thrown if a frequency is less than zero.

StepwiseRegression

```
public StepwiseRegression(double[,] cov, int nObservations)
```

Regression
Description
Creates a new instance of StepwiseRegression from a user-supplied variance-covariance matrix.

Parameters
- cov – A double matrix containing a variance-covariance or sum-of-squares and crossproducts matrix, in which the last column must correspond to the dependent variable.
- nObservations – An int containing the number of observations associated with cov.

Remarks
cov can be computed using the Imsl.Stat.Covariances (p. 545) class.

Methods

Compute
virtual public void Compute()

Description
Builds the multiple linear regression models using forward selection, backward selection, or stepwise selection.

Exceptions
- Imsl.Stat.NoVariables EnteredException is thrown if no variables entered the model. All elements of the Imsl.Stat.StepwiseRegression.ANOVA (p. 678) table are set to NaN
- Imsl.Stat.CyclingIsOccurringException is thrown if cycling occurs

Set Means
virtual public void SetMeans(double[] means)

Description
Sets the means of the variables.

Parameter
- means – A double array of length nVars+1, where nVars is the number of independent variables.
  means[0] through means[nVars-1] are the means of the independent variables and
  means[nVars] is the mean of the dependent variable.

Remarks
This is required when the covariance array is input and the intercept
Imsl.Stat.StepwiseRegression.Intercept (p. 680) is requested. Otherwise, it is not used.
Example: StepwiseRegression

This example uses a data set from Draper and Smith (1981, pp. 629-630). Method compute is invoked to find the best regression for each subset size using the $R^2$ criterion. By default, stepwise regression is performed.

```csharp
using System;
using Imsl;
using Imsl.Math;
using Imsl.Stat;

public class StepwiseRegressionEx1
{
    public static void Main(System.String[] args)
    {
        double[,] x = {{7.0, 26.0, 6.0, 60.0},
            {1.0, 29.0, 15.0, 52.0},
            {11.0, 56.0, 8.0, 20.0},
            {11.0, 31.0, 8.0, 47.0},
            {7.0, 52.0, 6.0, 33.0},
            {11.0, 55.0, 9.0, 22.0},
            {3.0, 71.0, 17.0, 6.0},
            {1.0, 31.0, 22.0, 44.0},
            {2.0, 54.0, 18.0, 22.0},
            {21.0, 47.0, 4.0, 26},
            {1.0, 40.0, 23.0, 34.0},
            {11.0, 66.0, 9.0, 12.0},
            {10.0, 68.0, 8.0, 12.0}};

        double[] y = new double[]{78.5, 74.3, 104.3, 87.6,
            95.9, 109.2, 102.7, 72.5,
            93.1, 115.9, 83.8, 113.3, 109.4};

        StepwiseRegression sr = new StepwiseRegression(x, y);
        sr.Compute();

        PrintMatrix pm = new PrintMatrix();
        PrintMatrixFormat pmf = new PrintMatrixFormat();
        pmf.NumberFormat = "0.000";
        pm.SetTitle("*** ANOVA *** "); pm.Print(sr.ANOVA.GetArray());

        StepwiseRegression.CoefficientTTestsValue coefT = sr.CoefficientTTests;
        double[,] coef = new double[4, 4];
        for (int i = 0; i < 4; i++)
        {
            coef[i,0] = coefT.GetCoefficient(i);
            coef[i,1] = coefT.GetStandardError(i);
            coef[i,2] = coefT.GetTStatistic(i);
            coef[i,3] = coefT.GetPValue(i);
        }
        pm.SetTitle("*** Coef *** "); pm.Print(pmf, coef);

        pm.SetTitle("*** Swept *** "); pm.Print(sr.Swept);
        pm.SetTitle("*** History *** "); pm.Print(sr.History);
        pm.SetTitle("*** VIF *** "); pm.Print(sr.CoefficientVIF);
    }
}
```
Output

*** ANOVA ***

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2641.00096476634</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>74.7621121567356</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2715.76307692308</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1320.50048238317</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7.47621121567356</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>176.626963081888</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1.58106023181439E-08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>97.2471047716931</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>96.6965257260317</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2.73426612012685</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>NaN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>NaN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*** Coef ***

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.440</td>
<td>0.138</td>
<td>10.403</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>0.416</td>
<td>0.186</td>
<td>2.242</td>
<td>0.052</td>
</tr>
<tr>
<td>2</td>
<td>-0.410</td>
<td>0.199</td>
<td>-2.058</td>
<td>0.070</td>
</tr>
<tr>
<td>3</td>
<td>-0.614</td>
<td>0.049</td>
<td>-12.621</td>
<td>0.000</td>
</tr>
</tbody>
</table>

*** Swept ***

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*** History ***

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*** VIF ***

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.0641052101769</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>18.7803086409578</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.45960147891528</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.0641052101769</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### StepwiseRegression.CoefficientTTestsValue Class

```java
public class Imsl.Stat.StepwiseRegression.CoefficientTTestsValue

CoefficientTTestsValue contains statistics related to the student-t test, for each regression coefficient.
```

#### Methods

**GetCoefficient**
```java
virtual public double GetCoefficient(int index)
```

**Description**
Returns the estimate for a coefficient of the independent variable.

**Parameter**
- `index` – An int which specifies the index of the coefficient whose estimate is to be returned.

**Returns**
A double which contains the estimate for the coefficient.

**Remarks**
`index` must be between 1 and the number of independent variables.

**GetPValue**
```java
virtual public double GetPValue(int index)
```

**Description**
Returns the $p$-value for the two-sided test $H_0 : \beta = 0$ vs. $H_1 : \beta \neq 0$.

**Parameter**
- `index` – An int which specifies the index of the coefficient whose $p$-value is to be returned.
Returns
A double which contains the estimated $p$-value for the coefficient.

Remarks
index must be between 1 and the number of independent variables.

GetStandardError
virtual public double GetStandardError(int index)

Description
Returns the estimated standard error for a coefficient estimate.

Parameter
    index – An int which specifies the index of the coefficient whose standard error estimate is to be returned.

Returns
A double which contains the estimated standard error for the coefficient.

Remarks
index must be between 1 and the number of independent variables.

GetTStatistic
virtual public double GetTStatistic(int index)

Description
Returns the student-$t$ test statistic for testing the $i$-th coefficient equal to zero ($\beta_{\text{index}} = 0$).

Parameter
    index – An int which specifies the index of the coefficient whose $t$-test statistic is to be returned.

Returns
A double which contains the estimated $t$-test statistic for the coefficient.

Remarks
index must be between 1 and the number of independent variables.

StepwiseRegression.Direction Enumeration

public enumeration Imsl.Stat.StepwiseRegression.Direction

Direction indicator.
### Fields

**Backward**

public Imsl.Stat.StepwiseRegression.Direction Backward

**Description**

Indicates backward regression. An attempt is made to remove a variable from the model. A variable is removed if its $p$-value exceeds $PValueOut$. During initialization, all candidate independent variables enter the model.

**Forward**

public Imsl.Stat.StepwiseRegression.Direction Forward

**Description**

Indicates forward regression. An attempt is made to add a variable to the model. A variable is added if its $p$-value is less than $PValueIn$. During initialization, only forced variables enter the model.

**Stepwise**

public Imsl.Stat.StepwiseRegression.Direction Stepwise

**Description**

Indicates stepwise regression. A backward step is attempted. After the backward step, a forward step is attempted. This is a stepwise step. Any forced variables enter the model during initialization.

### UserBasisRegression Class

public class Imsl.Stat.UserBasisRegression

Generates summary statistics using user-supplied functions in a nonlinear regression model.

Fits a linear function of the form

$$y = c_0 + c_1 f_1(x) + c_2 f_2(x) + \cdots + c_k f_k(x) + \varepsilon$$

where $f_1(x), f_2(x), \ldots, f_k(x)$ are the user basis functions $f_i(x)$ evaluated at index values $i = 1, 2, \ldots, k$. $c_0$ is the intercept, $c_1, c_2, \ldots, c_k$ are the coefficients associated with the basis functions, and $\varepsilon$ is the random error associated with $y$. The coefficients $c_0, c_1, \ldots, c_k$ are determined by least squares.

**Description**

UserBasisRegression generalizes the concept of linear regression to user defined basis functions. The linear regression model is

$$y = c_0 + c_1 x_1 + \cdots + c_k x_k + \varepsilon$$

where are the $k$ independent variables. UserBasisRegression generalizes this concept by setting $x_i = f_i(x)$, where $f_i(x)$ is any user defined function of $x$. 

Regression  UserBasisRegression • 689
This makes it easier for users to fit complex univariate models. For example, the LinearRegression class can be used to fit polynomials such as

\[ y = c_0 + c_1 x + c_2 x^2 \cdots + c_k x^k + \varepsilon \]

, but this requires an input matrix where the ith column of that array contains the values of \( x^i \).

With UserBasisRegression, these columns can be automatically generated. For this polynomial model, the user would define a user basis function \( f_i(x) = x^i \). The UserBasisRegression class automatically inserts the necessary values into the regression equation and then calculates the coefficients and analysis of variance statistics.

Since the user provides a method for calculating the basis function, other more complex user basis functions are possible such as

\[ y = c_1 \sin(x) + c_2 \cos(x) + \varepsilon \]

. In this case, \( nBasis=2 \), \( f_0(x) = \sin(x) \), and \( f_1(x) = \cos(x) \).

### Property

**ANOVA**

```csharp
public Imsl.Stat.ANOVA ANOVA {get; }
```

**Description**

An analysis of variance table and related statistics.

**Property Value**

An ANOVA table and related statistics.

### Constructor

**UserBasisRegression**

```csharp
public UserBasisRegression(Imsl.Stat.IRegressionBasis basis, int nBasis, bool hasIntercept)
```

**Description**

Constructs a UserBasisRegression object.

**Parameters**

- `basis` – A IRegressionBasis basis function supplied by the user.
- `nBasis` – A int which specifies the number of basis functions.
- `hasIntercept` – A bool which specifies whether or not the model has an intercept.
Methods

**GetCoefficients**

```java
public double[] GetCoefficients()
```

**Description**

Returns the regression coefficients.

**Returns**

A double array containing the regression coefficients.

**Remarks**

If `hasIntercept` is `false` its length is equal to the number of variables. If `hasIntercept` is `true` then its length is the number of variables plus one and the 0-th entry is the value of the intercept.

**Exception**

`Imsl.Math.SingularMatrixException` is thrown when the regression matrix is singular

**Update**

```java
public void Update(double x, double y, double w)
```

**Description**

Adds a new observation and associated weight to the `IRegressionBasis` object.

**Parameters**

- `x` – A double containing the independent (explanatory) variable.
- `y` – A double containing the dependent (response) variable.
- `w` – A double representing the weight.

**Example: Regression with User-supplied Basis Functions**

In this example, we fit the function $1 + \sin(x) + 7 \sin(3x)$ with no error introduced. The function is evaluated at 90 equally spaced points on the interval [0, 6]. Four basis functions are used, $\sin(kx)$ for $k = 1,...,4$ with no intercept.

```java
using System;
using Imsl.Stat;
using Imsl.Math;

public class UserBasisRegressionEx1 : IRegressionBasis
{
    public double Basis(int index, double x)
    {
        return System.Math.Sin((index + 1) * x);
    }

    public static void Main(string[] args)
    {
        double[] coef = new double[4];
    }

    public void Update(double x, double y, double w)
    {
        // Update code
    }
}
```
IRegressionBasis basis = new UserBasisRegressionEx1();
UserBasisRegression ubr =
    new UserBasisRegression(basis, 4, false);

for (int k = 0; k < 90; k++)
{
    double x = 6.0 * k / 89.0;
    double y = 1.0 + Math.Sin(x) + 7.0 * Math.Sin(3.0 * x);
    ubr.Update(x, y, 1.0);
}
coef = ubr.GetCoefficients();
PrintMatrix pm = new PrintMatrix("The regression coefficients are:");
PrintMatrixFormat pmf = new PrintMatrixFormat();
    pmf.NumberFormat = "0.000";
    pm.Print(pmf, coef);
}
}
}

Output
The regression coefficients are:
   0
  1 1.010
  2 0.020
  3 7.029
  4 0.037

Example: Regression with User-supplied Basis Functions

In this example, we fit the polynomial \( y = c_0 + c_1 x + c_2 x^2 + \cdots + c_4 x^4 + \varepsilon \). For this model, the user basis function is \( f_i(x) = x^{i+1} \) with \( i = 0, 1, \ldots, n\text{Basis} = 4 \) and hasIntercept=true.

Data are generated using the model \( y = 10 + 2x + 5x^3 \) with \( x = 0, 1, \ldots, 9 \).

using System;
using Imsl.Stat;
using Imsl.Math;

public class UserBasisRegressionEx2 : IRegressionBasis
{
    public double Basis(int index, double x)
    {
        // Notice zero-based indexing requires index be incremented
        return System.Math.Pow(x, index + 1);
    }

    public static void Main(string[] args)
    {
        double y = 0;
    double[] coef =
    IRegressionBasis basis = new UserBasisRegressionEx2();
    UserBasisRegression ubr =
        new UserBasisRegression(basis, 4, true);
for (double x = 0; x < 10; x++)
{
    y = 10.0 + x + 3 * System.Math.Pow(x, 3);
    ubr.Update(x, y, 1.0);
}
coef = ubr.GetCoefficients();
PrintMatrix pm = new PrintMatrix("The regression coefficients are: ");
PrintMatrixFormat pmf = new PrintMatrixFormat();
pmf.NumberFormat = "0.0";
pm.Print(pmf, coef);

// Print the Anova Table
ANOVA anovaTable = ubr.ANOVA;


Output
The regression coefficients are:
0
0 10.0
1 1.0
2 0.0
3 3.0

Regression UserBasisRegression • 693
Degrees Of Freedom For Model: 4
Degrees Of Freedom For Error: 5
Total (Corrected) Degrees Of Freedom: 9
Sum Of Squares For Model: 5152488
Sum Of Squares For Error: 1.86264514923096E-09
Total (Corrected) Sum Of Squares: 5152488
Model Mean Square: 1288122
Error Mean Square: 3.72529029846191E-10
F statistic: 3.45777616453632E+15
P value: 8.69662785564134E-39
R Squared (in percent): 100
Adjusted R Squared (in percent): 99.9999999999999
Model Error Standard deviation: 1.9305644846079E-06
Mean Of Y: 622
Coefficient Of Variation (in percent): 3.10305644846079E-06

IRegressionBasis Interface

public interface Imsl.Stat.IRegressionBasis

Interface for user supplied function to UserBasisRegression object.

Method

Basis

abstract public double Basis(int index, double x)

Description
Basis function for the nonlinear least-squares function.

Parameters

    index – A int which specifies the index of the basis function to be evaluated at x.
    x – A double which specifies the point at which the function is to be evaluated.

Returns
A double which specifies the returned value of the function at x.
Chapter 14: Analysis of Variance

Types

class ANCOVA ................................................................. 695
class ANOVA ................................................................. 708
enumeration ANOVA.ComputeOption .................................. 718
class ANOVAFactorial ......................................................... 719
enumeration ANOVAFactorial.ErrorCalculation ...................... 729
class MultipleComparisons .................................................. 730

Usage Notes

The classes described in this chapter are for commonly-used experimental designs. Typically, responses are stored in the input vector \( y \) in a pattern that takes advantage of the balanced design structure. Consequently, the full set of model subscripts is not needed to identify each response. The classes assume the usual pattern, which requires that the last model subscript change most rapidly, followed by the model subscript next in line, and so forth, with the first subscript changing at the slowest rate. This pattern is referred to as lexicographical ordering.

ANOVA class allows missing responses if confidence interval information is not requested. Double.NaN (Not a Number) is the missing value code used by these classes. Any element of \( y \) that is missing must be set to NaN. Other classes described in this chapter do not allow missing responses because the classes generally deal with balanced designs.

As a diagnostic tool for determination of the validity of a model, classes in this chapter typically perform a test for lack of fit when \( n(n > 1) \) responses are available in each cell of the experimental design.

**ANCOVA Class**

public class Imsl.Stat.ANCOVA

Analyzes a one-way classification model with covariates.
Class ANCOVA performs analyses for models that combine the features of a one-way analysis of variance model with that of a multiple linear regression model. The basic one-way analysis of covariance model is

\[
y_{ij} = \beta_0 + \beta_1 x_{i1j} + \beta_2 x_{i2j} + \ldots + \beta_m x_{ijm} + \epsilon_{ij}
\]

\[
i = 1, 2, \ldots, n_{\text{group}}
\]
\[
j = 1, 2, \ldots, n_i
\]

where, \(n_{\text{group}}\) is the number of treatment groups, the observed value of \(y_{ij}\) constitutes the \(j\)-th response in the \(i\)-th group, \(\beta_0\) denotes the \(y\) intercept for the regression function for the \(i\)-th group, \(\beta_1, \beta_2, \ldots, \beta_m\) are the regression coefficients for the covariates, and the \(\epsilon_{ij}\)'s are independently distributed normal errors with mean zero and variance \(\sigma^2\). This model allows the regression function for each group to have different intercepts. However, the remaining \(m\) regression coefficients are the same for each group, i.e., the regression functions are parallel.

In practice, sometimes the regression functions are not parallel. In addition to estimates for the model assuming parallelism (parallel regression planes), ANCOVA computes estimates and summary statistics for the separate regressions of each group. These estimates can be examined using the methods GetCoefficientTables and GetANOVATables.

Estimates for the \(\beta_0\)'s and \(\beta_1, \beta_2, \ldots, \beta_m\) in the model assuming parallelism are returned using the method GetModelCoefficients. Summary statistics are also computed for this model and returned by the Compute method.

The adjusted group means, stored in the last column of \(xymean\), are computed using the formula:

\[
\hat{\beta}_0 + \hat{\beta}_1 \bar{x}_1 + \hat{\beta}_2 \bar{x}_2 + \ldots + \hat{\beta}_m \bar{x}_m
\]

where \(xymean\) is the matrix returned by GetMeans and \(n_{\text{cov}}\) is the number of covariates.

The estimated covariance between the \(i_1\)-th and \(i_2\)-th adjusted group mean is given by

\[
\nu_{i_1 i_2} + \sum_{r=1}^{m} \sum_{s=1}^{m} \bar{x}_r \nu_{r+k+s} + \sum_{r=1}^{m} \bar{x}_s \nu_{1, k+r} + \sum_{r=1}^{m} \bar{x}_r \nu_{2, k+r}
\]

where \(\nu_{pq}\) is the entry in \(\text{covb}[p-1][q-1]\), where \(\text{covb}\) is returned by GetVarCovCoefficients and is the estimated covariance between the \(p\)-th and \(q\)-th estimated coefficients in the regression function.

### Property

**NumberOfMissing**

virtual public int NumberOfMissing {get; }  

**Description**

The number of cases with missing values in covariates or responses. Cases with any missing values are not used in the analysis.
Property Value
An int scalar value indicating the number of cases with missing values in covariates or responses.

Constructor

**ANCOVA**

public ANCOVA(double[][] responses, double[][][] covariates)

Description
Constructs a one-way classification model with covariates.

Parameters
- **responses** – A double matrix containing the responses. Each row in responses corresponds to a treatment group. Each row of responses can contain a different number of observations. There must be at least two groups (responses.Length > 1).
- **covariates** – A three-dimensional double array containing the covariates. The first dimension corresponds to the number of covariates (consider each element an individual covariate matrix or covariates.Length = number of covariates). Each row in covariates[i] corresponds to a treatment group. There must be the same number of rows, for each covariate, as there are in the responses matrix (covariates[i].Length = responses.Length). There must be at least one covariate (covariate.Length > 1). Each row of covariates[i] must contain the same number of elements as the corresponding row in responses (covariates[i][j].Length = responses[j].Length).

Methods

**Compute**

public double[] Compute()

Description
Performs one-way analysis of covariance assuming parallelism and returns an array containing the parallelism tests for the one-way analysis of covariance.

Returns
A double array containing the parallelism tests for the one-way analysis of covariance organized as follows:
### GetAdjustedANOVA

**Description**

Returns the partial sum of squares for the one-way analysis of covariance.

**Returns**

A double array containing the partial sum of squares for the one-way analysis of covariance organized as follows:

<table>
<thead>
<tr>
<th>index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Degrees of freedom for groups after covariates.</td>
</tr>
<tr>
<td>1</td>
<td>Degrees of freedom for covariates after groups.</td>
</tr>
<tr>
<td>2</td>
<td>Sum of squares for groups after covariates.</td>
</tr>
<tr>
<td>3</td>
<td>Sum of squares for model (groups and covariates combined).</td>
</tr>
<tr>
<td>4</td>
<td>$F$-statistic for groups.</td>
</tr>
<tr>
<td>5</td>
<td>$F$-statistic for covariates.</td>
</tr>
<tr>
<td>6</td>
<td>$p$-value for groups.</td>
</tr>
<tr>
<td>7</td>
<td>$p$-value for covariates.</td>
</tr>
</tbody>
</table>

### GetANCOVA

**Description**

Returns an array containing the one-way analysis of covariance assuming parallelism.

**Returns**

A double array of length 15 containing the following values:
GetANOVATables

public double[][] GetANOVATables()

Description

Returns a matrix of size ngroup by 15 containing the analysis of variance tables for each linear regression model fitted separately to each treatment group.

Returns

A double matrix containing the analysis of variance tables for each linear regression model fitted separately to each treatment group. The 15 values in the i-th row are for treatment group i organized as follows:

<table>
<thead>
<tr>
<th>index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Degrees of freedom for model</td>
</tr>
<tr>
<td>1</td>
<td>Degrees of freedom for error</td>
</tr>
<tr>
<td>2</td>
<td>Total degrees of freedom</td>
</tr>
<tr>
<td>3</td>
<td>Sum of squares for model</td>
</tr>
<tr>
<td>4</td>
<td>Sum of squares for error</td>
</tr>
<tr>
<td>5</td>
<td>Total sum of squares</td>
</tr>
<tr>
<td>6</td>
<td>Model mean square</td>
</tr>
<tr>
<td>7</td>
<td>Error mean square</td>
</tr>
<tr>
<td>8</td>
<td>F statistic</td>
</tr>
<tr>
<td>9</td>
<td>p-value</td>
</tr>
<tr>
<td>10</td>
<td>R-squared (in percent)</td>
</tr>
<tr>
<td>11</td>
<td>Adjusted R-squared (in percent)</td>
</tr>
<tr>
<td>12</td>
<td>Estimated standard deviation of the model error</td>
</tr>
<tr>
<td>13</td>
<td>Mean of the response (dependent variable)</td>
</tr>
<tr>
<td>14</td>
<td>Coefficient of variation (in percent)</td>
</tr>
</tbody>
</table>

Analysis of Variance ANCOVA • 699
**GetCoefficientTable**

public double[][] GetCoefficientTable(int group)

**Description**

Returns a matrix of size $ncov + 1$ by 4 containing statistics for a linear regression model fitted separately for each of the $ngroup$ treatment groups.

**Parameter**

- group – An int specifying the group for which the regression statistics will be retrieved.

**Returns**

A double matrix containing statistics for a group. Each row corresponds to the model coefficients. For row = 0, the statistics relate to the intercept in the regression model. For row = 1, 2, ..., $ncov$, the statistics relate to the slopes for the covariates. The column dimension corresponds to the row described for GetModelCoefficients as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Coefficient estimate.</td>
</tr>
<tr>
<td>1</td>
<td>Estimated standard error</td>
</tr>
<tr>
<td>2</td>
<td>$t$-statistic.</td>
</tr>
<tr>
<td>3</td>
<td>$p$-value.</td>
</tr>
</tbody>
</table>

**GetCoefficientTables**

public double[][][] GetCoefficientTables()

**Description**

Returns an array containing statistics for a linear regression model fitted separately for all $ngroup$ treatments.

**Returns**

A double[][][] array containing statistics for a linear regression model fitted separately for each of the $ngroup$ treatment groups. The 3 dimensional array organized with $ngroup$ rows, $ncov + 1$ columns, and depth of 4. Each row corresponds to one of the $ngroup$ treatment groups. Each column corresponds to the model coefficients. For column = 0, the statistics relate to the intercept in the regression model. For column = 1, 2, ..., $ncov$, the statistics relate to the slopes for the covariates. The depth dimension corresponds to the columns described for GetModelCoefficients as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Coefficient estimate.</td>
</tr>
<tr>
<td>1</td>
<td>Estimated standard error</td>
</tr>
<tr>
<td>2</td>
<td>$t$-statistic.</td>
</tr>
<tr>
<td>3</td>
<td>$p$-value.</td>
</tr>
</tbody>
</table>

**GetMeans**

public double[][] GetMeans()
Description

Returns a matrix containing the unadjusted means for the covariates and the response variate and the means for the response variate adjusted for the covariates.

Returns

A double matrix of size $ngroup + 1$ by $ncov + 3$ containing the unadjusted means for the covariates and the response variate and the means for the response variate adjusted for the covariates. Each row for $i = 0, 1, ..., ngroup - 1$ corresponds to a group. Row $ngroup$ contains overall statistics. The means are organized in columns as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Number of non-missing cases</td>
</tr>
<tr>
<td>1 thru $ncov$</td>
<td>Covariate means.</td>
</tr>
<tr>
<td>$ncov + 1$</td>
<td>Response mean.</td>
</tr>
<tr>
<td>$ncov + 2$</td>
<td>Response mean adjusted assuming parallelism.</td>
</tr>
</tbody>
</table>

GetModelCoefficients

public double[][] GetModelCoefficients()

Description

Returns a matrix containing statistics for the regression coefficients for the model assuming parallelism.

Returns

A double matrix of size $ngroup + ncov$ by 4 containing statistics for the regression coefficients for the model assuming parallelism. Each row corresponds to a coefficient in the model. For $i = 0, 1, ..., ngroup - 1$, row $i$ is for the y intercept for the $i$-th group. The remaining $ncov$ rows are for the covariate coefficients. The statistics in the columns are organized as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Coefficient estimate.</td>
</tr>
<tr>
<td>1</td>
<td>Estimated standard error of the estimate.</td>
</tr>
<tr>
<td>2</td>
<td>$t$-statistic.</td>
</tr>
<tr>
<td>3</td>
<td>$p$-value.</td>
</tr>
</tbody>
</table>

GetR

public double[][] GetR()

Description

Returns the $R$ matrix from the $QR$ decomposition.

Returns

A double matrix of size $ngroup + ncov$ by $ngroup + ncov$ which contains the $R$ from the $QR$ decomposition.
Remarks
The \( R \) matrix is from the regression assuming parallelism.

GetVarCovAdjustedMeans
public double[][] GetVarCovAdjustedMeans()

Description
Returns a matrix containing the estimated variances and covariances for the adjusted means assuming parallelism.

Returns
A double matrix of size \( n_{\text{group}} \) by \( n_{\text{group}} \) containing the estimated variances and covariances for the adjusted means assuming parallelism.

GetVarCovCoefficients
public double[][] GetVarCovCoefficients()

Description
Returns a matrix containing the estimated variances and covariances for the coefficients returned using GetModelCoefficients.

Returns
A matrix of size \( n_{\text{group}} + n_{\text{cov}} \) by \( n_{\text{group}} + n_{\text{cov}} \) containing the estimated variances and covariances for the coefficients returned using GetModelCoefficients.

Example 1: One-way analysis of covariance model
This example fits a one-way analysis of covariance model assuming parallelism using data discussed by Snedecor and Cochran (Table 14.6.1, pages 432-436). The responses are concentrations of cholesterol (in mg/100 ml) in the blood of two groups of women: women from Iowa and women from Nebraska. The age of a woman is the single covariate. The cholesterol concentrations and ages of the women according to state are shown in the following table. (There are 11 Iowa women and 19 Nebraska women in the study. Only the first 5 women from each state are shown here.)

<table>
<thead>
<tr>
<th>Iowa</th>
<th>Nebraska</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Cholesterol</td>
</tr>
<tr>
<td>46</td>
<td>181</td>
</tr>
<tr>
<td>52</td>
<td>228</td>
</tr>
<tr>
<td>39</td>
<td>182</td>
</tr>
<tr>
<td>65</td>
<td>249</td>
</tr>
<tr>
<td>54</td>
<td>259</td>
</tr>
</tbody>
</table>

There is no evidence from the data to indicate that the regression lines for cholesterol concentration as a function of age are not parallel for Iowa and Nebraska women (p-value is 0.5425). The parallel line model suggests that Nebraska women may have higher cholesterol concentrations than Iowa women. The cholesterol concentrations (adjusted for age) are 195.5 for Iowa women versus 224.2 for Nebraska women. The difference is 28.7 with an estimated standard error of

\[ \sqrt{170.4 + 97.4 - 2(2.9)} = 16.1 \]
using System;
using Imsl.Stat;
using Imsl.Math;

public class ANCOVAEx1
{
    public static void Main(String[] args)
    {
        double[][] y = new double[][]{
            new double[]{181.0, 228.0, 182.0, 249.0, 259.0, 201.0, 121.0,
                339.0, 224.0, 112.0, 189.0},
            new double[]{137.0, 173.0, 177.0, 241.0, 225.0, 223.0, 190.0,
                257.0, 337.0, 189.0, 214.0, 140.0, 196.0, 262.0,
                261.0, 356.0, 159.0, 191.0, 197.0}};
        double[][] x = new double[1][][ ];
        for (int i = 0; i < 1; i++)
        {
            x[i] = new double[2][];
        }
        x[0][0] = new double[]{46.0, 52.0, 39.0, 65.0, 54.0, 33.0, 49.0, 76.0,
            71.0, 41.0, 58.0};
        x[0][1] = new double[]{18.0, 44.0, 33.0, 78.0, 51.0, 43.0, 44.0, 63.0,
            19.0, 42.0, 30.0, 47.0, 58.0, 70.0,
            67.0, 31.0, 21.0, 56.0};
        ANCOVA awc = new ANCOVA(y, x);
        double[] testpl = awc.Compute();
        double[] aov = awc.GetANCOVA();
        Console.WriteLine(" * * * Analysis of Variance * * *");
        Console.WriteLine(" Sum of Mean " +
            " Prob of");
        Console.WriteLine("Source DF Squares Square Overall F Larger F");
        Console.WriteLine("Model {0,3:0.#} {1,10:0.##} {2,9:0.##} {3,2:0.##} {4,8:0.######} ", aov[0], aov[3], aov[6], aov[8],
            aov[9]);
        Console.WriteLine("Error {0,3:0.#} {1,10:0.##} {2,7:0.##} ",
            aov[1], aov[4], aov[7]);
        Console.WriteLine("Total {0,3:0.#} {1,10:0.##} \n\n" +
            " Prob of");
        Console.WriteLine("Source DF Squares Square Test Larger F");
        Console.WriteLine("Nonparallelism {0,3:0.#} {1,10:0.##} {2,7:0.##} {3,7:0.#####} {4,5:0.####} ",
            testpl[0], testpl[3], testpl[6], testpl[8], testpl[9]);
        Console.WriteLine("Extra Assuming");
        Console.WriteLine("Nonparallelism {0,3:0.#} {1,10:0.##} {2,7:0.##} ",
            testpl[1], testpl[4], testpl[7]);
        Console.WriteLine("Error Assuming");
        Console.WriteLine("Parallelism {0,3:0.#} {1,10:0.##} \n\n");
testpl[2], testpl[5]);
PrintMatrixFormat pmf = new Imsl.Math.PrintMatrixFormat();

pmf.NumberFormat = "0.###";
new PrintMatrix("XY Mean Matrix\n").Print(pmf, awc.GetMeans());
new PrintMatrix("\nVar./Covar. Matrix of Adjusted Group Means\n").Print(pmf, awc.GetVarCovAdjustedMeans());

Output

* * * Analysis of Variance * * *

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sums of Squares</th>
<th>Mean Square</th>
<th>Overall F</th>
<th>Prob of Larger F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>2</td>
<td>54432.75</td>
<td>27216.38</td>
<td>14.97</td>
<td>0.000042</td>
</tr>
<tr>
<td>Error</td>
<td>27</td>
<td>49103.91</td>
<td>1818.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>29</td>
<td>103536.67</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* * * Test for Parallelism * * *

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sums of Squares</th>
<th>Mean Square</th>
<th>Test</th>
<th>Prob of Larger F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra due to Nonparallelism</td>
<td>1</td>
<td>709.05</td>
<td>709.05</td>
<td>0.38093</td>
<td>0.5425</td>
</tr>
<tr>
<td>Extra Assuming Nonparallelism</td>
<td>26</td>
<td>48394.86</td>
<td>1861.34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Assuming Parallelism</td>
<td>27</td>
<td>49103.91</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XY Mean Matrix

0 1 2 3
0 11 53.091 207.727 195.521
1 19 45.947 217.105 224.172
2 30 48.567 213.667 213.667

Var./Covar. Matrix of Adjusted Group Means

0 1
0 -170.368 -2.915
1 -2.915 97.407

Example 2: One-way analysis of covariance model

This example fits a one-way analysis of covariance model and performs a test for parallelism using data discussed by Snedecor and Cochran (1967, Table 14.8.1, pages 438-443). The responses are weight gains (in pounds per day) of 40 pigs for four groups of pigs under varying treatments. Two covariates—initial age (in days) and initial weight (in pounds)—are used. For each treatment, there are 10
pigs. Only the first 5 pigs from each treatment are shown here.

<table>
<thead>
<tr>
<th>Treatment 1</th>
<th>Treatment 2</th>
<th>Treatment 3</th>
<th>Treatment 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Wt.</td>
<td>Gain</td>
<td>Age</td>
</tr>
<tr>
<td>78</td>
<td>61</td>
<td>1.4</td>
<td>78</td>
</tr>
<tr>
<td>90</td>
<td>59</td>
<td>1.79</td>
<td>99</td>
</tr>
<tr>
<td>94</td>
<td>76</td>
<td>1.72</td>
<td>80</td>
</tr>
<tr>
<td>71</td>
<td>50</td>
<td>1.47</td>
<td>75</td>
</tr>
<tr>
<td>99</td>
<td>61</td>
<td>1.26</td>
<td>94</td>
</tr>
</tbody>
</table>

For these data, the test for non-parallelism is not statistically significant ($p = 0.901$). The one-way analysis of covariance test for the treatment means adjusted for the covariates, assuming parallel slopes, is statistically significant at a stated significance level of $\alpha = 0.05$, ($p = 0.04931$). Multiple comparisons can be done using the least significant difference approach of comparing each pair of treatment groups with the two-sample student-t test. Since the adjusted means in the one-way analysis of covariance are correlated, the standard error for these comparisons must be computed using the variances and covariances in covm. The standard errors for these comparisons are fairly similar ranging from 0.0630 to 0.0638. The Student’s t comparisons identify differences between groups 1 and 2, and 1 and 4 as being statistically significant with p-values of 0.01225 and 0.03854 respectively.

```java
using System;
using Imsl.Stat;
using Imsl.Math;

class ANCOVAEx2
{
    public static void Main(String[] args)
    {
        int j;
        int ngroup = 4;
        double[][] x1 = new double[][]{
            new double[]{78.0, 90.0, 94.0, 71.0, 99.0, 80.0, 83.0, 75.0, 62.0, 67.0},
            new double[]{78.0, 99.0, 80.0, 75.0, 94.0, 91.0, 75.0, 63.0, 62.0, 67.0},
            new double[]{78.0, 83.0, 79.0, 70.0, 85.0, 83.0, 71.0, 66.0, 67.0, 67.0},
            new double[]{77.0, 71.0, 78.0, 70.0, 95.0, 96.0, 71.0, 63.0, 62.0, 67.0}};
        double[][] x2 = new double[][]{
            new double[]{61.0, 59.0, 76.0, 50.0, 61.0, 54.0, 57.0, 45.0, 41.0, 40.0},
            new double[]{74.0, 75.0, 64.0, 48.0, 62.0, 42.0, 52.0, 43.0, 50.0, 40.0},
            new double[]{80.0, 61.0, 62.0, 47.0, 59.0, 42.0, 47.0, 42.0, 40.0, 40.0},
            new double[]{62.0, 55.0, 62.0, 43.0, 57.0, 51.0, 41.0, 40.0, 45.0, 39.0}};
        double[][] y = new double[][]{
            new double[]{1.40, 1.79, 1.72, 1.47, 1.26, 1.28, 1.34, 1.55, 1.57, 1.26},
            new double[]{1.61, 1.31, 1.12, 1.35, 1.29, 1.24, 1.29, 1.43, 1.57, 1.26}};
```
new double[]{1.67, 1.41, 1.73, 1.23, 1.49, 1.22, 1.39, 1.39, 1.56, 1.36},
new double[]{1.40, 1.47, 1.37, 1.15, 1.22, 1.48, 1.31, 1.27, 1.22, 1.36});
double[][] x = new double[][]{{x1, x2};
/* setup covariate input matrix */
ANCOVA awc = new ANCOVA(y, x);
double[] testpl = awc.Compute();
double[] aov = awc.GetANCOVA();
double[] adjAov = awc.GetAdjustedANOVA();
double[] xymean = awc.GetMeans();
double[] covm = awc.GetVarCovAdjustedMeans();
Console.WriteLine(" * * * Test for Parallelism * * *");
Console.WriteLine(" Sum of Mean " + "F Prob of");
Console.WriteLine(" Source DF Squares Square " + "Test Larger F");
Console.WriteLine("Extra due to");
Console.WriteLine("Nonparallelism {0,3:0.#} {1,10:0.##} " +
"{2,7:0.#####} {3,7:0.#####} {4,5:0.###}", testpl[0], testpl[3],
testpl[6], testpl[8], testpl[9]);
Console.WriteLine("Nonparallelism {0,3:0.#} {1,10:0.##} {2,7:0.##}",
testpl[1], testpl[4], testpl[7]);
Console.WriteLine("Error Assuming");
Console.WriteLine("Parallelism {0,3:0.#} {1,10:0.##} 

", testpl[2], testpl[5]);
Console.WriteLine(" * * * Analysis of Variance * * *");
Console.WriteLine(" Sum of " + "Overall F Larger F");
Console.WriteLine("Model {0,3:0.#} {1,7:0.#####} {2,7:0.#####} {3,7:0.#####} " +
"{4,5:0.#####} {5,7:0.#####} {6,7:0.#####} {7,7:0.#####}", aov[0], aov[3], aov[6], aov[8], aov[9]);
Console.WriteLine("Error {0,3:0.#} {1,7:0.#####} {2,7:0.#####} ", aov[1],
aov[4], aov[7]);
Console.WriteLine("Total {0,3:0.#} {1,7:0.#####} 

", aov[2], aov[5]);
Console.WriteLine(" * * * Adjusted Analysis of Variance * " +
"* #\n");
Console.WriteLine(" Sum of " +
"F Prob of");
Console.WriteLine("Source DF Squares " +
"Test Larger F");
Console.WriteLine("Groups after Covariates {0,3:0.#} {1,10:0.##} " +
"{2,5:0.##} {3,7:0.#####} {4,5:0.###}, adjAov[0], adjAov[2],
adjAov[4], adjAov[6]);
Console.WriteLine("Covariates after Groups {0,3:0.#} {1,10:0.##} " +
"{2,5:0.##} {3,7:0.#####}\n", adjAov[1], adjAov[3],
adjAov[5], adjAov[7]);
Console.WriteLine(" * * * Group Means * * *");
for (int i = 0; i < ngroup; i++)
{ double se = Math.Sqrt(covm[i][i]);

}
```csharp
Console.WriteLine("{0} | {1,5:0.####} | {2,5:0.####} | {3,7:0.####} | {4,7:0.#####}", i + 1, xymean[i][ngroup - 1], xymean[i][ngroup], se);
}
Console.WriteLine("\n\n* * * Student-t Multiple Comparisons * " + "* *");
Console.WriteLine(" Groups | Diff | Std. Error | Student-t | P-Value" + "*
");
for (int i = 0; i < ngroup - 1; i++)
{
    for (j = i + 1; j < ngroup; j++)
    {
        double delta = xymean[i][ngroup] - xymean[j][ngroup];
        double se = Math.Sqrt(covm[i][i] + covm[j][j] - 2.0 * covm[i][j]);
        double t = delta / se;
        double df = xymean[i][0] + xymean[j][0] - 2;
        double pvalue = 1.0 - Cdf.StudentsT(t, df);
        Console.WriteLine("{0} vs {1} | {2,7:0.####} | {3,7:0.####} | {4,7:0.####} | {5,7:0.#####}", i + 1, j + 1, delta, se, t, pvalue);
    }
}

Output

* * * Test for Parallelism * * *

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>Mean Square</th>
<th>F Test</th>
<th>Prob of Larger F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra due to Nonparallelism</td>
<td>6</td>
<td>0.05</td>
<td>0.01</td>
<td>0.35534</td>
<td>0.901</td>
</tr>
<tr>
<td>Extra Assuming Nonparallelism</td>
<td>28</td>
<td>0.62</td>
<td>0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Assuming Parallelism</td>
<td>34</td>
<td>0.67</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* * * Analysis of Variance * * *

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>Mean Square</th>
<th>Overall F</th>
<th>Prob of Larger F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>5</td>
<td>0.352517</td>
<td>0.070503</td>
<td>3.57639</td>
<td>0.0105</td>
</tr>
<tr>
<td>Error</td>
<td>34</td>
<td>0.670261</td>
<td>0.019714</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>39</td>
<td>1.022778</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* * * Adjusted Analysis of Variance * * *

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>F Test</th>
<th>Prob of Larger F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groups after Covariates</td>
<td>3</td>
<td>0.17</td>
<td>2.9</td>
<td>0.04931</td>
</tr>
<tr>
<td>Covariates after Groups</td>
<td>2</td>
<td>0.17</td>
<td>4.44</td>
<td>0.01939</td>
</tr>
</tbody>
</table>

Analysis of Variance

ANCOVA • 707
**Group Means**

<table>
<thead>
<tr>
<th>GROUP</th>
<th>Unadjusted</th>
<th>Adjusted</th>
<th>Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.464</td>
<td>1.4614</td>
<td>0.0448</td>
</tr>
<tr>
<td>2</td>
<td>1.319</td>
<td>1.3068</td>
<td>0.0446</td>
</tr>
<tr>
<td>3</td>
<td>1.445</td>
<td>1.4429</td>
<td>0.0447</td>
</tr>
<tr>
<td>4</td>
<td>1.325</td>
<td>1.3418</td>
<td>0.0449</td>
</tr>
</tbody>
</table>

**Student-t Multiple Comparisons**

<table>
<thead>
<tr>
<th>Groups</th>
<th>Diff</th>
<th>Std. Error</th>
<th>Student-t</th>
<th>P-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 vs 2</td>
<td>0.1546</td>
<td>0.063</td>
<td>2.4548</td>
<td>0.01225</td>
</tr>
<tr>
<td>1 vs 3</td>
<td>0.0185</td>
<td>0.0637</td>
<td>0.2902</td>
<td>0.3875</td>
</tr>
<tr>
<td>1 vs 4</td>
<td>0.1196</td>
<td>0.0638</td>
<td>1.8752</td>
<td>0.03854</td>
</tr>
<tr>
<td>2 vs 3</td>
<td>-0.1362</td>
<td>0.0632</td>
<td>-2.1528</td>
<td>0.97743</td>
</tr>
<tr>
<td>2 vs 4</td>
<td>-0.035</td>
<td>0.0638</td>
<td>-0.5495</td>
<td>0.70528</td>
</tr>
<tr>
<td>3 vs 4</td>
<td>0.1011</td>
<td>0.0631</td>
<td>1.6018</td>
<td>0.0633</td>
</tr>
</tbody>
</table>

**ANOVA Class**

```csharp
public class Imsl.Stat.ANOVA
    Analysis of Variance table and related statistics.

Properties

**AdjustedRSquared**

```csharp
public double AdjustedRSquared {get; }
```

**Description**

Returns the adjusted R-squared (in percent).

**Property Value**

A double representing the adjusted R-squared (in percent).

**CoefficientOfVariation**

```csharp
public double CoefficientOfVariation {get; }
```

**Description**

Returns the coefficient of variation (in percent).
Property Value
A double representing the coefficient of variation (in percent).

DegreesOfFreedomForError
public double DegreesOfFreedomForError {get; }

Description
Returns the degrees of freedom for error.

Property Value
A double representing the degrees of freedom for the error.

DegreesOfFreedomForModel
public double DegreesOfFreedomForModel {get; }

Description
Returns the degrees of freedom for the model.

Property Value
A double representing the degrees of freedom for the model.

ErrorMeanSquare
public double ErrorMeanSquare {get; }

Description
Returns the error mean square.

Property Value
A double representing the error mean square.

F
public double F {get; }

Description
Returns the F statistic.

Property Value
A double representing the F statistic

MeanOfY
public double MeanOfY {get; }

Description
Returns the mean of the response (dependent variable).

Property Value
A double representing the mean of the response (dependent variable).

ModelErrorStdev
public double ModelErrorStdev {get; }
Description
Returns the estimated standard deviation of the model error.

Property Value
A double representing the estimated standard deviation of the model error.

ModelMeanSquare
default double ModelMeanSquare {get; }

Description
Returns the model mean square.

Property Value
A double representing the model mean square.

NumberOfProcessors
public int NumberOfProcessors {get; set; }

Description
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

Property Value
An int indicating the maximum possible number of processors to use.

Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

P
public double P {get; }

Description
Returns the p-value.

Property Value
A double representing the p-value.

RSquared
public double RSquared {get; }

Description
Returns the R-squared (in percent).
Property Value
A double representing the $R^2$ (in percent).

**SumOfSquaresForError**
public double SumOfSquaresForError {get; }

Description
Returns the sum of squares for error.

Property Value
A double representing the sum of squares for the error.

**SumOfSquaresForModel**
public double SumOfSquaresForModel {get; }

Description
Returns the sum of squares for model.

Property Value
A double representing the sum of squares for the model.

**TotalDegreesOfFreedom**
public double TotalDegreesOfFreedom {get; }

Description
Returns the total degrees of freedom.

Property Value
A double representing the total degrees of freedom.

**TotalMissing**
public int TotalMissing {get; }

Description
Returns the total number of missing values.

Property Value
An int representing the total number of missing values (NaN) in input Y.

Remarks
Elements of Y containing NaN (not a number) are omitted from the computations.

**TotalSumOfSquares**
public double TotalSumOfSquares {get; }

Description
Returns the total sum of squares.

Analysis of Variance ANOVA • 711
Property Value
A double representing the total sum of squares.

Constructors

ANOVA
public ANOVA(double[][] y)

Description
Analyzes a one-way classification model.

Parameter
  y – Two-dimension double array containing the responses.

Remarks
The rows in y correspond to observation groups. Each row of y can contain a different number of observations.

ANOVA
public ANOVA(double dfr, double ssr, double dfe, double sse, double gmean)

Description
Construct an analysis of variance table and related statistics. Intended for use by the LinearRegression class.

Parameters
  dfr – A double representing the degrees of freedom for the model.
  ssr – A double representing the sum of squares for the model.
  dfe – A double representing the degrees of freedom for the error.
  sse – A double representing the sum of squares for the error.
  gmean – A double representing the grand mean.

Remarks
If the grand mean is not known it may be set to not-a-number.

Methods

GetArray
public double[] GetArray()
Description
Returns the ANOVA values as an array.

Returns
A double[15] array containing the following values.

<table>
<thead>
<tr>
<th>index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Degrees of freedom for model</td>
</tr>
<tr>
<td>1</td>
<td>Degrees of freedom for error</td>
</tr>
<tr>
<td>2</td>
<td>Total degrees of freedom</td>
</tr>
<tr>
<td>3</td>
<td>Sum of squares for model</td>
</tr>
<tr>
<td>4</td>
<td>Sum of squares for error</td>
</tr>
<tr>
<td>5</td>
<td>Total sum of squares</td>
</tr>
<tr>
<td>6</td>
<td>Model mean square</td>
</tr>
<tr>
<td>7</td>
<td>Error mean square</td>
</tr>
<tr>
<td>8</td>
<td>F statistic</td>
</tr>
<tr>
<td>9</td>
<td>p-value</td>
</tr>
<tr>
<td>10</td>
<td>R-squared (in percent)</td>
</tr>
<tr>
<td>11</td>
<td>Adjusted R-squared (in percent)</td>
</tr>
<tr>
<td>12</td>
<td>Estimated standard deviation of the model error</td>
</tr>
<tr>
<td>13</td>
<td>Mean of the response (dependent variable)</td>
</tr>
<tr>
<td>14</td>
<td>Coefficient of variation (in percent)</td>
</tr>
</tbody>
</table>

GetConfidenceInterval
virtual public double[] GetConfidenceInterval(double conLevel, int i, int j, Imsl.Stat.ANOVA.ComputeOption compMethod)

Description
Computes the confidence interval associated with the difference of means between two groups using a specified method.

Parameters
conLevel – A double specifying the confidence level for simultaneous interval estimation. If the Tukey method for computing the confidence intervals on the pairwise difference of means is to be used, conLevel must be in the range [90.0, 99.0]. Otherwise, conLevel must be in the range [0.0, 100.0). One normally sets this value to 95.0.
i – An int indicating the i-th member of the pair difference, \( \mu_i - \mu_j \). i must be a valid group index.
j – An int indicating the j-th member of the pair difference, \( \mu_i - \mu_j \). j must be a valid group index.
compMethod – An ANOVA.ComputeOption. compMethod must be one of the following:
**compMethod** | **Description**
--- | ---
Tukey | Uses the Tukey method. This method is valid for balanced one-way designs.
TukeyKramer | Uses the Tukey-Kramer method. This method simplifies to the Tukey method for the balanced case.
DunnSidak | Uses the Dunn-Sidak method.
Bonferroni | Uses the Bonferroni method.
Scheffe | Uses the Scheffe method.
OneAtATime | Uses the One-at-a-Time (Fisher’s LSD) method.

**Returns**

A double array containing the group numbers, difference of means, and lower and upper confidence limits.

| Array Element | Description |
--- | ---|
0 | Group number for the $i$-th mean. |
1 | Group number for the $j$-th mean. |
2 | Difference of means ($i$-th mean) - ($j$-th mean). |
3 | Lower confidence limit for the difference. |
4 | Upper confidence limit for the difference. |

**Remarks**

GetConfidenceInterval computes the simultaneous confidence interval on the pairwise comparison of means $\mu_i$ and $\mu_j$ in the one-way analysis of variance model. Any of several methods can be chosen. A good review of these methods is given by Stoline (1981). Also the methods are discussed in many elementary statistics texts, e.g., Kirk (1982, pages 114-127). Let $s^2$ be the estimated variance of a single observation. Let $v$ be the degrees of freedom associated with $s^2$. Let

$$\alpha = 1 - \frac{conLevel}{100.0}$$

The methods are summarized as follows:

**Tukey method**: The Tukey method gives the narrowest simultaneous confidence intervals for the pairwise differences of means $\mu_i - \mu_j$ in balanced ($n_1 = n_2 = \ldots = n_k = n$) one-way designs. The method is exact and uses the Studentized range distribution. The formula for the difference $\mu_i - \mu_j$ is given by

$$\bar{y}_i - \bar{y}_j \pm q_{1-\alpha,k,v} \sqrt{\frac{s^2}{n}}$$

where $q_{1-\alpha,k,v}$ is the $(1 - \alpha)100$ percentage point of the Studentized range distribution with parameters $k$ and $v$. If the group sizes are unequal, the Tukey-Kramer method is used instead.

**Tukey-Kramer method**: The Tukey-Kramer method is an approximate extension of the Tukey method for the unbalanced case. (The method simplifies to the Tukey method for the balanced case.) The method always produces confidence intervals narrower than the Dunn-Sidak and Bonferroni methods. Hayter (1984) proved that the method is conservative, i.e., the method guarantees a confidence coverage of at least $(1 - \alpha)100\%$. Hayter’s proof gave further support to earlier recommendations for its use.
(Stoline 1981). (Methods that are currently better are restricted to special cases and only offer improvement in severely unbalanced cases, see, e.g., Spurrier and Isham 1985). The formula for the difference $\mu_i - \mu_j$ is given by the following:

$$\bar{y}_i - \bar{y}_j \pm q_{1-\alpha, k} \sqrt{\frac{s^2}{2n_i} + \frac{s^2}{2n_j}}$$

**Dunn-Sidak method:** The Dunn-Sidak method is a conservative method. The method gives wider intervals than the Tukey-Kramer method. (For large $\nu$ and small $\alpha$ and $k$, the difference is only slight.) The method is slightly better than the Bonferroni method and is based on an improved Bonferroni (multiplicative) inequality (Miller, pages 101, 254-255). The method uses the $t$ distribution. The formula for the difference $\mu_i - \mu_j$ is given by

$$\bar{y}_i - \bar{y}_j \pm t_{1+\frac{1}{2}, \frac{n}{n_i}+\frac{n}{n_j}} \sqrt{\frac{s^2}{n_i} + \frac{s^2}{n_j}}$$

where $t_{f, \nu}$ is the $100f$ percentage point of the $t$ distribution with $\nu$ degrees of freedom.

**Bonferroni method:** The Bonferroni method is a conservative method based on the Bonferroni (additive) inequality (Miller, page 8). The method uses the $t$ distribution. The formula for the difference $\mu_i - \mu_j$ is given by

$$\bar{y}_i - \bar{y}_j \pm t_{1, \frac{n}{n_i}+\frac{n}{n_j}} \sqrt{\frac{s^2}{n_i} + \frac{s^2}{n_j}}$$

**Scheffé method:** The Scheffé method is an overly conservative method for simultaneous confidence intervals on pairwise difference of means. The method is applicable for simultaneous confidence intervals on all contrasts, i.e., all linear combinations

$$\sum_{i=1}^{k} c_i \mu_i$$

where the following is true:

$$\sum_{i=1}^{k} c_i = 0$$

The method can be recommended here only if a large number of confidence intervals on contrasts in addition to the pairwise differences of means are to be constructed. The method uses the $F$ distribution. The formula for the difference $\mu_i - \mu_j$ is given by

$$\bar{y}_i - \bar{y}_j \pm (k-1) F_{1, k-1, \nu} \sqrt{\frac{s^2}{n_i} + \frac{s^2}{n_j}}$$

where $F_{1, k-1, \nu}$ is the $(1 - \alpha)$ 100 percentage point of the $F$ distribution with $k - 1$ and $\nu$ degrees of freedom.

**One-at-a-time $t$ method (Fisher’s LSD):** The one-at-a-time $t$ method is the method appropriate for constructing a single confidence interval. The confidence percentage input is appropriate for one interval.
at a time. The method has been used widely in conjunction with the overall test of the null hypothesis \( \mu_1 = \mu_2 = \ldots = \mu_k \) by the use of the F statistic. Fisher’s LSD (least significant difference) test is a two-stage test that proceeds to make pairwise comparisons of means only if the overall F test is significant. Milliken and Johnson (1984, page 31) recommend LSD comparisons after a significant F only if the number of comparisons is small and the comparisons were planned prior to the analysis. If many unplanned comparisons are made, they recommend Scheffe’s method. If the F test is insignificant, a few planned comparisons for differences in means can still be performed by using either Tukey, Tukey-Kramer, Dunn-Sidak or Bonferroni methods. Because the F test is insignificant, Scheffe’s method will not yield any significant differences. The formula for the difference \( \mu_i - \mu_j \) is given by

$$
\bar{y}_i - \bar{y}_j \pm t_{1-\frac{\alpha}{2}} \sqrt{\frac{s^2}{n_i} + \frac{s^2}{n_j}}
$$

### GetGroupInformation

```csharp
public double[][] GetGroupInformation()
```

**Description**

Returns information concerning the groups.

**Returns**

A two-dimensional double array containing information concerning the groups.

**Remarks**

Row i contains information pertaining to the i-th group. The information in the columns is as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Group Number</td>
</tr>
<tr>
<td>1</td>
<td>Number of nonmissing observations</td>
</tr>
<tr>
<td>2</td>
<td>Group Mean</td>
</tr>
<tr>
<td>3</td>
<td>Group Standard Deviation</td>
</tr>
</tbody>
</table>

### Example: ANOVA

This example computes a one-way analysis of variance for data discussed by Searle (1971, Table 5.1, pages 165-179). The responses are plant weights for 6 plants of 3 different types - 3 normal, 2 off-types, and 1 aberrant. The 3 normal plant weights are 101, 105, and 94. The 2 off-type plant weights are 84 and 88. The 1 aberrant plant weight is 32. Note in the results that for the group with only one response, the standard deviation is undefined and is set to NaN (not a number).

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;

public class ANOVAEEx1
{
    public static void Main(String[] args)
```
double[][] y = {
    new double[]{101, 105, 94},
    new double[]{84, 88},
    new double[]{32}};

ANOVA anova = new ANOVA(y);
double[] aov = anova.GetArray();

Console.WriteLine("Degrees Of Freedom For Model = " + aov[0]);
Console.WriteLine("Degrees Of Freedom For Error = " + aov[1]);
Console.WriteLine("Total (Corrected) Degrees Of Freedom = " + aov[2]);
Console.WriteLine("Sum Of Squares For Model = " + aov[3]);
Console.WriteLine("Sum Of Squares For Error = " + aov[4]);
Console.WriteLine("Total (Corrected) Sum Of Squares = " + aov[5]);
Console.WriteLine("Model Mean Square = " + aov[6]);
Console.WriteLine("Error Mean Square = " + aov[7]);
Console.WriteLine("F statistic = " + aov[8]);
Console.WriteLine("P value= " + aov[9]);
Console.WriteLine("R Squared (in percent) = " + aov[10]);
Console.WriteLine("Adjusted R Squared (in percent) = " + aov[11]);
Console.WriteLine("Model Error Standard deviation = " + aov[12]);
Console.WriteLine("Mean Of Y = " + aov[13]);
Console.WriteLine("Coefficient Of Variation (in percent) = " + aov[14]);
Console.WriteLine("Total number of missing values = " + anova.TotalMissing);

Output

Degrees Of Freedom For Model = 2
Degrees Of Freedom For Error = 3
Total (Corrected) Degrees Of Freedom = 5
Sum Of Squares For Model = 3480
Sum Of Squares For Error = 70
Total (Corrected) Sum Of Squares = 3550
Model Mean Square = 1740
Error Mean Square = 23.3333333333333
F statistic = 74.5714285714286
P value= 0.00276888252534978
R Squared (in percent) = 98.0281690140845

Analysis of Variance

ANOVA • 717
Adjusted R Squared (in percent) = 96.7136150234742
Model Error Standard deviation = 4.83045891539648
Mean Of Y = 84
Coefficient Of Variation (in percent) = 5.75054632785295
Total number of missing values = 0

Group Information

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>100</td>
<td>5.56776436283002</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>86</td>
<td>2.82842712474619</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>32</td>
<td>NaN</td>
</tr>
</tbody>
</table>

ANOVA.ComputeOption Enumeration

public enumeration Imsl.Stat.ANOVA.ComputeOption

Compute option.

Fields

Bonferroni

public Imsl.Stat.ANOVA.ComputeOption Bonferroni

Description
The Bonferroni method.

DunnSidak

public Imsl.Stat.ANOVA.ComputeOption DunnSidak

Description
The Dunn-Sidak method.

OneAtATime

public Imsl.Stat.ANOVA.ComputeOption OneAtATime

Description
The One-at-a-Time (Fisher’s LSD) method.

Scheffe

public Imsl.Stat.ANOVA.ComputeOption Scheffe
The Scheffe method.

Tukey

public Imsl.Stat.ANOVA.ComputeOption Tukey

Description

The Tukey method.

TukeyKramer

public Imsl.Stat.ANOVA.ComputeOption TukeyKramer

Description

The Tukey-Kramer method.

ANOVAFactorial Class

public class Imsl.Stat.ANOVAFactorial

Analyzes a balanced factorial design with fixed effects.

Class ANOVAFactorial performs an analysis for an n-way classification design with balanced data. For balanced data, there must be an equal number of responses in each cell of the n-way layout. The effects are assumed to be fixed effects. The model is an extension of the two-way model to include n factors. The interactions (two-way, three-way, up to n-way) can be included in the model, or some of the higher-way interactions can be pooled into error. The ModelOrder property specifies the number of factors to be included in the highest-way interaction. For example, if three-way and higher-way interactions are to be pooled into error, set ModelOrder = 2. (By default, ModelOrder = nSubscripts - 1 with the last subscript being the error subscript.) Pure indicates there are repeated responses within the n-way cell; Pooled indicates otherwise.

Class ANOVAFactorial requires the responses as input into a single vector y in lexicographical order, so that the response subscript associated with the first factor varies least rapidly, followed by the subscript associated with the second factor, and so forth. Hemmerle (1967, Chapter 5) discusses the computational method.

Properties

ErrorIncludeType

public Imsl.Stat.ANOVAFactorial.ErrorCalculation ErrorIncludeType {get; set; }
Description
The error included type.

Property Value
An int scalar specifying the included error type.

Remarks
ANOVAFactorial.ErrorCalculation.Pure, the default option, indicates factor nSubscripts is error. Its main effect and all its interaction effects are pooled into the error with the other (ModelOrder + 1)-way and higher-way interactions.

ANOVAFactorial.ErrorCalculation.Pooled indicates factor nSubscripts is not error. Only (ModelOrder + 1)-way and higher-way interactions are included in the error.

modelOrder
public int ModelOrder {get; set; }

Description
The number of factors to be included in the highest-way interaction in the model.

Property Value
An int scalar containing the number of factors to be included in the highest-way interaction in the model.

Remarks
ModelOrder must be in the interval [1, nSubscripts-1]. For example:
ModelOrder of 1 indicates that a main effect model will be analyzed.
ModelOrder of 2 indicates that two-way interactions will be included in the model.
By default, ModelOrder = nSubscripts - 1.

Constructor

ANOVAFactorial
public ANOVAFactorial(int nSubscripts, int[] nLevels, double[] y)

Description
Constructor for ANOVAFactorial.

Parameters
nSubscripts – An int scalar containing the number of subscripts. Number of factors in the model + 1 (for the error term).
nLevels – An int array of length nSubscripts containing the number of levels for each of the factors for the first nSubscripts-1 elements. nLevels[nSubscripts-1] is the number of observations per cell.
y – A double array of length nLevels[0] * nLevels[1] * ... * nLevels[nSubscripts-1] containing the responses.
**Remarks**

y must not contain NaN for any of its elements; i.e., missing values are not allowed.

**Exception**

System.ArgumentException is thrown if nLevels.length, and y.length are not consistent

---

**Methods**

**Compute**

```java
public double Compute()
```

**Description**

Analyzes a balanced factorial design with fixed effects.

**Returns**

A double scalar containing the p-value for the overall F test.

**GetANOVATable**

```java
public double[] GetANOVATable()
```

**Description**

Returns the analysis of variance table. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

**Returns**

A double array containing the analysis of variance table.

**Remarks**

The analysis of variance statistics are given as follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Analysis of Variance Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Degrees of freedom for the model</td>
</tr>
<tr>
<td>1</td>
<td>Degrees of freedom for error</td>
</tr>
<tr>
<td>2</td>
<td>Total (corrected) degrees of freedom</td>
</tr>
<tr>
<td>3</td>
<td>Sum of squares for the model</td>
</tr>
<tr>
<td>4</td>
<td>Sum of squares for error</td>
</tr>
<tr>
<td>5</td>
<td>Total (corrected) sum of squares</td>
</tr>
<tr>
<td>6</td>
<td>Model mean square</td>
</tr>
<tr>
<td>7</td>
<td>Error mean square</td>
</tr>
<tr>
<td>8</td>
<td>Overall F-statistic</td>
</tr>
<tr>
<td>9</td>
<td>p-value</td>
</tr>
<tr>
<td>10</td>
<td>R^2 (in percent)</td>
</tr>
<tr>
<td>11</td>
<td>Adjusted R^2 (in percent)</td>
</tr>
<tr>
<td>12</td>
<td>Estimate of the standard deviation</td>
</tr>
<tr>
<td>13</td>
<td>Overall mean of y</td>
</tr>
<tr>
<td>14</td>
<td>Coefficient of variation (in percent)</td>
</tr>
</tbody>
</table>
GetMeans
public double[] GetMeans()

Description
Returns the subgroup means. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns
A double array containing the subgroup means.

GetTestEffects
public double[,] GetTestEffects()

Description
Returns statistics relating to the sums of squares for the effects in the model. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns
A double matrix containing statistics relating to the sums of squares for the effects in the model.

Remarks
Here,

\[ \text{NEF} = \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{\min(n, \text{model order})} \]

where \( n \) is given by nSubscripts if ANOVAFactorial.ErrorCalculation.Pooled is specified; otherwise, nSubscripts-1. Suppose the factors are A, B, C, and error. With ModelOrder = 3, rows 0 through NEF-1 would correspond to A, B, C, and error. With ModelOrder = 3, rows 0 through NEF-1 would correspond to A, B, C, AB, AC, BC, and ABC, respectively.

The columns of the output matrix are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>1</td>
<td>Sum of squares</td>
</tr>
<tr>
<td>2</td>
<td>( F )-statistic</td>
</tr>
<tr>
<td>3</td>
<td>( p )-value</td>
</tr>
</tbody>
</table>

Example 1: Two-way Analysis of Variance

A two-way analysis of variance is performed with balanced data discussed by Snedecor and Cochran (1967, Table 12.5.1, p. 347). The responses are the weight gains (in grams) of rats that were fed diets varying in the source (A) and level (B) of protein. The model is

\[ y_{ijk} = \mu + \alpha_i + \beta_j + \gamma_{ij} + \epsilon_{ijk} \quad i = 1, 2; \quad j = 1, 2, 3; \quad k = 1, 2, \ldots, 10 \]
where

$$\sum_{i=1}^{2} \alpha_i = 0; \sum_{j=1}^{3} \beta_j = 0; \sum_{i=1}^{2} \gamma_{ij} = 0 \text{ for } j = 1, 2, 3;$$

and

$$\sum_{j=1}^{3} \gamma_{ij} = 0 \text{ for } j = 1, 2$$

The first responses in each cell in the two-way layout are given in the following table:

<table>
<thead>
<tr>
<th>Protein Level (B)</th>
<th>Protein Source (A)</th>
<th>Beef</th>
<th>Cereal</th>
<th>Pork</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Beef</td>
<td>73, 102, 118, 104, 81, 107, 100, 87, 117, 111</td>
<td>98, 74, 56, 111, 95, 88, 82, 77, 86, 92</td>
<td>94, 79, 96, 98, 102, 102, 108, 91, 120, 105</td>
</tr>
<tr>
<td>Low</td>
<td>Beef</td>
<td>90, 76, 90, 64, 86, 51, 72, 90, 95, 78</td>
<td>107, 95, 97, 80, 98, 74, 74, 67, 89, 58</td>
<td>49, 82, 73, 86, 81, 97, 106, 70, 61, 82</td>
</tr>
<tr>
<td></td>
<td>Cereal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pork</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class ANOVAFactorialEx1
{
    public static void Main(String[] args)
    {
        int nSubscripts = 3;
        int[] nLevels = new int[] {3, 2, 10};
        double[] y = new double[] {
            73.0, 102.0, 118.0, 104.0, 81.0, 107.0, 100.0, 87.0, 117.0, 111.0, 90.0, 64.0, 86.0, 51.0, 72.0, 90.0, 95.0, 78.0, 95.0, 88.0, 98.0, 74.0, 56.0, 111.0, 95.0, 80.0, 98.0, 77.0, 86.0, 92.0, 107.0, 95.0, 97.0, 80.0, 98.0, 74.0, 74.0, 67.0, 89.0, 58.0, 94.0, 79.0, 96.0, 98.0, 102.0, 102.0, 108.0, 91.0, 120.0, 105.0, 49.0, 82.0, 73.0, 86.0, 81.0, 97.0, 106.0, 70.0, 61.0, 82.0};
    }
}
ANOVAFactorial af =
    new ANOVAFactorial(nSubscripts, nLevels, y);
Console.Out.WriteLine
    ("P-value = " + af.Compute().ToString("0.000000");
}
}

Output
P-value = 0.002299

Example 2: Two-way Analysis of Variance

In this example, the same model and data is fit as in the example 1, but additional information is printed.

using System;
using Imsl.Stat;

public class ANOVAFactorialEx2
{
    public static void Main(String[] args)
    {
        int nSubscripts = 3, i;
        int[] nLevels = new int[]{3, 2, 10};
        double[] y = new double[]{ 73.0, 102.0, 118.0,
            104.0, 81.0, 107.0,
            100.0, 87.0, 117.0,
            111.0, 90.0, 76.0,
            90.0, 64.0, 86.0,
            51.0, 72.0, 90.0,
            95.0, 78.0, 98.0,
            74.0, 56.0, 111.0,
            95.0, 88.0, 82.0,
            77.0, 86.0, 92.0,
            107.0, 95.0, 97.0,
            80.0, 98.0, 74.0,
            74.0, 67.0, 89.0,
            58.0, 94.0, 79.0,
            96.0, 98.0, 102.0,
            102.0, 108.0, 91.0,
            120.0, 105.0, 49.0,
            82.0, 73.0, 86.0,
            81.0, 97.0, 106.0,
            70.0, 61.0, 82.0};

        String[] labels =
            new String[] {"degrees of freedom for the model",
                "degrees of freedom for error",
                "total (corrected) degrees of freedom",
                "sum of squares for the model",
                "sum of squares for error",
                "total (corrected) sum of squares",
                "model mean square"};

        af = new ANOVAFactorial(nSubscripts, nLevels, y);

        String out = af.Compute().ToString("0.000000") + " 
            P-value = 
            Degrees of freedom for the model = 
            Degrees of freedom for error = 
            Total (corrected) degrees of freedom = 
            Sum of squares for the model = 
            Sum of squares for error = 
            Total (corrected) sum of squares = 
            Model mean square = " + labels.Length;
        String in = af.Compute().ToString("0.000000") + " 
            P-value = 
            Degrees of freedom for the model = 
            Degrees of freedom for error = 
            Total (corrected) degrees of freedom = 
            Sum of squares for the model = 
            Sum of squares for error = 
            Total (corrected) sum of squares = 
            Model mean square = " + labels.Length;
ANOVAFactorial af = new ANOVAFactorial(nSubscripts, nLevels, y);
Output

P-value = 0.002299

* * * Analysis of Variance * * *

degrees of freedom for the model 5.0000
degrees of freedom for error 54.0000
total (corrected) degrees of freedom 59.0000
sum of squares for the model 4612.9333
sum of squares for error 11586.0000
total (corrected) sum of squares 16198.9333
model mean square 922.5867
error mean square 214.5556
F-statistic 4.3000
p-value 0.0023
R-squared (in percent) 28.4768
Adjusted R-squared (in percent) 21.8543
est. standard deviation of the model error 14.6477
overall mean of y 87.8667
coefficient of variation (in percent) 16.6704

* * * Variation Due to the Model * * *

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>Mean Square</th>
<th>Prob. of Larger F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2.0000</td>
<td>266.5333</td>
<td>0.6211</td>
<td>0.5411</td>
</tr>
<tr>
<td>B</td>
<td>1.0000</td>
<td>3168.2667</td>
<td>14.7666</td>
<td>0.0003</td>
</tr>
<tr>
<td>A*B</td>
<td>2.0000</td>
<td>1178.1333</td>
<td>2.7455</td>
<td>0.0732</td>
</tr>
</tbody>
</table>

* * * Subgroup Means * * *

| grand mean | 87.8667 |
| A1 | 89.6000 |
| A2 | 84.9000 |
| A3 | 89.1000 |
| B1 | 95.1333 |
| B2 | 80.6000 |
| A1*B1 | 100.0000 |
| A2*B1 | 85.9000 |
| A2*B2 | 83.9000 |
| A3*B1 | 99.5000 |
| A3*B2 | 78.7000 |

Example 3: Three-way Analysis of Variance

This example performs a three-way analysis of variance using data discussed by John (1971, pp. 91 92). The responses are weights (in grams) of roots of carrots grown with varying amounts of applied nitrogen (A), potassium (B), and phosphorus (C). Each cell of the three-way layout has one response. Note that the ABC interactions sum of squares, which is 186, is given incorrectly by John (1971, Table 5.2.) The three-way layout is given in the following table:
### Analysis of Variance

#### ANOVA Factorial Ex3

```csharp
using System;
using Imsl.Stat;

class ANOVAFactorialEx3
{
    public static void Main(String[] args)
    {
        int nSubscripts = 3, i;
        int[] nLevels = new int[]{3, 3, 3};
        double[] y = new double[]{ 88.76, 87.45, 86.01, 91.41, 98.27, 104.2, 97.85, 96.85, 90.09, 94.83, 84.57, 81.06, 100.49, 97.2, 120.8, 99.75, 112.3, 108.77, 99.9, 92.98, 94.72, 100.23, 107.77, 118.39, 104.51, 110.94, 102.87};
        String[] labels =
            new String[]{"degrees of freedom for the model" + ",
                "degrees of freedom for error" + 
                "total (corrected) degrees of freedom",
                "sum of squares for the model",
                "sum of squares for error",
                "total (corrected) sum of squares",
                "model mean square",
                "error mean square",
                "F-statistic",
                "p-value",
                "R-squared (in percent)",
                "Adjusted R-squared (in percent)",
                "est. standard deviation of the model error",
                "overall mean of y",
                "coefficient of variation (in percent)"};
        String[] rlabels =
        ANOVAFactorial af =
            new ANOVAFactorial(nSubscripts, nLevels, y);
    }
}
```
af.ErrorIncludeType = ANOVAFactorial.ErrorCalculation.Pooled;
Console.Out.WriteLine("P-value = " + af.Compute().ToString("0.000000");

double[] anova = af.GetANOVATable();
for (i = 0; i < anova.Length; i++)
{
    Console.Out.WriteLine(labels[i] + " " + anova[i].ToString("0.0000"));
}

int len = sb.Length;
for (int j = 0; j < (8 - len); j++)
    sb.Append(' ');
len = sb.Length;
for (int j = 0; j < (11 - len); j++)
    sb.Append(' ');
len = sb.Length;
for (int j = 0; j < (17 - len); j++)
    sb.Append(' ');
len = sb.Length;
for (int j = 0; j < (23 - len); j++)
    sb.Append(' ');
len = sb.Length;
for (int j = 0; j < (31 - len); j++)
    sb.Append(' ');
len = sb.Length;
for (int j = 0; j < (39 - len); j++)
    sb.Append(' ');
len = sb.Length;
for (int j = 0; j < (46 - len); j++)
    sb.Append(' ');

Console.Out.WriteLine(sb.ToString());

Output

P-value = 0.008299

* * * Analysis of Variance * * *
degrees of freedom for the model 18.0000

728 • ANOVAFactorial IMSL C# Numerical Library
degrees of freedom for error 8.0000
total (corrected) degrees of freedom 26.0000
sum of squares for the model 2395.7290
sum of squares for error 185.7763
total (corrected) sum of squares 2581.5052
model mean square 133.0961
error mean square 23.2220
F-statistic 5.7315
p-value 0.0083
R-squared (in percent) 92.8036
Adjusted R-squared (in percent) 76.6116
est. standard deviation of the model error 4.8189
overall mean of \( y \) 98.9619
coefficient of variation (in percent) 4.8695

* * * Variation Due to the Model * * *

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>Mean Square</th>
<th>Prob. of Larger F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2.0000</td>
<td>488.3675</td>
<td>10.5152</td>
<td>0.0058</td>
</tr>
<tr>
<td>B</td>
<td>2.0000</td>
<td>1090.6564</td>
<td>23.4832</td>
<td>0.0004</td>
</tr>
<tr>
<td>C</td>
<td>2.0000</td>
<td>49.1485</td>
<td>1.0582</td>
<td>0.3911</td>
</tr>
<tr>
<td>A*B</td>
<td>4.0000</td>
<td>142.5853</td>
<td>1.5350</td>
<td>0.2804</td>
</tr>
<tr>
<td>A*C</td>
<td>4.0000</td>
<td>32.3474</td>
<td>0.3482</td>
<td>0.8383</td>
</tr>
<tr>
<td>B*C</td>
<td>4.0000</td>
<td>592.6238</td>
<td>6.3800</td>
<td>0.0131</td>
</tr>
</tbody>
</table>

**ANOVAFactorial.ErrorCalculation** Enumeration

public enumeration Imsl.Stat.ANOVAFactorial.ErrorCalculation

ErrorCalculation members indicate whether interaction effects are pooled into the error or not.

**Fields**

**Pooled**

public Imsl.Stat.ANOVAFactorial.ErrorCalculation Pooled

**Description**

Indicates factor nSubscripts is not error.

**Pure**

public Imsl.Stat.ANOVAFactorial.ErrorCalculation Pure

**Description**

Indicates factor nSubscripts is error. This is the default.
MultipleComparisons Class

public class Imsl.Stat.MultipleComparisons

Performs Student-Newman-Keuls multiple comparisons test.

Class MultipleComparisons performs a multiple comparison analysis of means using the
Student-Newman-Keuls method. The null hypothesis is equality of all possible ordered subsets of a set
of means. This null hypothesis is tested using the Studentized range of each of the corresponding subsets
of sample means. The method is discussed in many elementary statistics texts, e.g., Kirk (1982, pp.
123-125).

Property

Alpha

public double Alpha {get; set; }

Description

The significance level of the test

Property Value

A double scalar containing the significance level of test.

Remarks

Alpha must be in the interval [0.01, 0.10]. By default, Alpha = 0.01.

Constructor

MultipleComparisons

public MultipleComparisons(double[] means, int df, double stdError)

Description

Constructor for MultipleComparisons.

Parameters

means – A double array containing the means.

df – A int scalar containing the degrees of freedom associated with stdError.

stdError – A double scalar containing the effective estimated standard error of a mean.
**Remarks**

In fixed effects models, stdError equals the estimated standard error of a mean. For example, in a one-way model \( \text{stdError} = \sqrt{s^2/n} \) where \( s^2 \) is the estimate of \( \sigma^2 \) and \( n \) is the number of responses in a sample mean. In models with random components, use \( \text{stdError} = \text{sedif}/\sqrt{2} \) where \( \text{sedif} \) is the estimated standard error of the difference of two means.

**Method**

**Compute**

```csharp
public int[] Compute()
```

**Description**

Performs Student-Newman-Keuls multiple comparisons test.

**Returns**

A `int` array, call it `equalMeans`, indicating the size of the groups of means declared to be equal.

**Remarks**

Value `equalMeans[I] = J` indicates the \( I \)-th smallest mean and the next \( J-1 \) larger means are declared equal. Value `equalMeans[I] = 0` indicates no group of means starts with the \( I \)-th smallest mean.

**Example: Multiple Comparisons Test**

A multiple-comparisons analysis is performed using data discussed by Kirk (1982, pp. 123-125). The results show that there are three groups of means with three separate sets of values: (36.7, 40.3, 43.4), (40.3, 43.4, 47.2), and (43.4, 47.2, 48.7).

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class MultipleComparisonsEx1
{
    public static void Main(String[] args)
    {
        double[] means = new double[]{36.7, 48.7, 43.4, 47.2, 40.3};

        /* Perform multiple comparisons tests */
        MultipleComparisons mc =
            new MultipleComparisons(means, 45, 1.6970563);

        new PrintMatrix("Size of Groups of Means").Print(mc.Compute());
    }
}
```
Output

Size of Groups of Means

0
0 3
1 3
2 3
3 0
Chapter 15: Categorical and Discrete Data Analysis

Types

class ContingencyTable ................................................................. 733
class CategoricalGenLinModel ......................................................... 747
enumeration CategoricalGenLinModel.DistributionParameterModel ........... 770

Usage Notes

The ContingencyTable class computes many statistics of interest in a two-way table. Statistics computed by this routine include the usual chi-squared statistics, measures of association, Kappa, and many others.

ContingencyTable Class

public class Imsl.Stat.ContingencyTable

Performs a chi-squared analysis of a two-way contingency table.

Class ContingencyTable computes statistics associated with an \( r \times c \) contingency table. The function computes the chi-squared test of independence, expected values, contributions to chi-squared, row and column marginal totals, some measures of association, correlation, prediction, uncertainty, the McNemar test for symmetry, a test for linear trend, the odds and the log odds ratio, and the kappa statistic (if the appropriate optional arguments are selected).

Notation

Let \( x_{ij} \) denote the observed cell frequency in the \( ij \) cell of the table and \( n \) denote the total count in the table. Let \( p_{ij} = p_i \cdot p_j \cdot \) denote the predicted cell probabilities under the null hypothesis of independence, where \( p_i \cdot \) and \( p_j \cdot \) are the row and column marginal relative frequencies. Next, compute the expected
cell counts as \( e_{ij} = np_{ij} \).

Also required in the following are \( a_{uv} \) and \( b_{uv} \) for \( u, v = 1, \ldots, n \). Let \( (r_s, c_s) \) denote the row and column response of observation \( s \). Then, \( a_{uv} = 1, 0, \) or \(-1\), depending on whether \( r_u < r_v, r_u = r_v, \) or \( r_u > r_v \), respectively. The \( b_{uv} \) are similarly defined in terms of the \( c_s \) variables.

**Chi-squared Statistic**

For each cell in the table, the contribution to \( \chi^2 \) is given as \((x_{ij} - e_{ij})^2/e_{ij}\). The Pearson chi-squared statistic (denoted \( \chi^2 \)) is computed as the sum of the cell contributions to chi-squared. It has \((r - 1)(c - 1)\) degrees of freedom and tests the null hypothesis of independence, i.e., \( H_0 : p_{ij} = p_i \cdot p_j \cdot \). The null hypothesis is rejected if the computed value of \( \chi^2 \) is too large.

The maximum likelihood equivalent of \( \chi^2 \), \( G^2 \) is computed as follows:

\[
G^2 = -2 \sum_{i,j} x_{ij} \ln(x_{ij}/np_{ij})
\]

\( G^2 \) is asymptotically equivalent to \( \chi^2 \) and tests the same hypothesis with the same degrees of freedom.

**Measures Related to Chi-squared (Phi, Contingency Coefficient, and Cramer’s V)**

There are three measures related to chi-squared that do not depend on sample size:

\[
\text{phi, } \phi = \sqrt{\frac{\chi^2}{n}}
\]

\[
\text{contingency coefficient, } P = \sqrt{\frac{\chi^2}{n + \chi^2}}
\]

\[
\text{Cramer’s V, } V = \sqrt{\frac{\chi^2}{n \min(r,c)}}
\]

Since these statistics do not depend on sample size and are large when the hypothesis of independence is rejected, they can be thought of as measures of association and can be compared across tables with different sized samples. While both \( P \) and \( V \) have a range between 0.0 and 1.0, the upper bound of \( P \) is actually somewhat less than 1.0 for any given table (see Kendall and Stuart 1979, p. 587). The significance of all three statistics is the same as that of the \( \chi^2 \) statistic, which is contained in the \( \text{ChiSquared} \) property.

The distribution of the \( \chi^2 \) statistic in finite samples approximates a chi-squared distribution. To compute the exact mean and standard deviation of the \( \chi^2 \) statistic, Haldane (1939) uses the multinomial distribution with fixed table marginals. The exact mean and standard deviation generally differ little from the mean and standard deviation of the associated chi-squared distribution.

**Standard Errors and p-values for Some Measures of Association**
In Columns 1 through 4 of statistics, estimated standard errors and asymptotic \( p \)-values are reported. Estimates of the standard errors are computed in two ways. The first estimate, in Column 1 of the return matrix from the \texttt{Statistics} property, is asymptotically valid for any value of the statistic. The second estimate, in Column 2 of the array, is only correct under the null hypothesis of no association. The \( z \)-scores in Column 3 of statistics are computed using this second estimate of the standard errors. The \( p \)-values in Column 4 are computed from this \( z \)-score. See Brown and Benedetti (1977) for a discussion and formulas for the standard errors in Column 2.

**Measures of Association for Ranked Rows and Columns**

The measures of association, \( \phi \), \( P \), and \( V \), do not require any ordering of the row and column categories. Class \texttt{ContingencyTable} also computes several measures of association for tables in which the rows and column categories correspond to ranked observations. Two of these tests, the product-moment correlation and the Spearman correlation, are correlation coefficients computed using assigned scores for the row and column categories. The cell indices are used for the product-moment correlation, while the average of the tied ranks of the row and column marginals is used for the Spearman rank correlation. Other scores are possible.

Gamma, Kendall’s \( \tau_b \), Stuart’s \( \tau_c \), and Somers’ \( D \) are measures of association that are computed like a correlation coefficient in the numerator. In all these measures, the numerator is computed as the “covariance” between the \( a_{uv} \) variables and \( b_{uv} \) variables defined above, i.e., as follows:

\[
\sum_u \sum_v a_{uv} b_{uv}
\]

Recall that \( a_{uv} \) and \( b_{uv} \) can take values -1, 0, or 1. Since the product \( a_{uv} b_{uv} = 1 \) only if \( a_{uv} \) and \( b_{uv} \) are both 1 or are both -1, it is easy to show that this “covariance” is twice the total number of agreements minus the number of disagreements, where a disagreement occurs when \( a_{uv} b_{uv} = -1 \).

Kendall’s \( \tau_b \) is computed as the correlation between the \( a_{uv} \) variables and the \( b_{uv} \) variables (see Kendall and Stuart 1979, p. 593). In a rectangular table \((r \neq c)\), Kendall’s \( \tau_b \) cannot be 1.0 (if all marginal totals are positive). For this reason, Stuart suggested a modification to the denominator of \( \tau \) in which the denominator becomes the largest possible value of the “covariance.” This maximizing value is approximately \( n^2m/(m-1) \), where \( m = \min (r, c) \). Stuart’s \( \tau_c \) uses this approximate value in its denominator. For large \( n \), \( \tau_c \approx m \tau_b / (m-1) \).

Gamma can be motivated in a slightly different manner. Because the “covariance” of the \( a_{uv} \) variables and the \( b_{uv} \) variables can be thought of as twice the number of agreements minus the disagreements, \( 2(A - D) \), where \( A \) is the number of agreements and \( D \) is the number of disagreements, Gamma is motivated as the probability of agreement minus the probability of disagreement, given that either agreement or disagreement occurred. This is shown as \( \gamma = (A - D) / (A + D) \).

Two definitions of Somers’ \( D \) are possible, one for rows and a second for columns. Somers’ \( D \) for rows can be thought of as the regression coefficient for predicting \( a_{uv} \) from \( b_{uv} \). Moreover, Somer’s \( D \) for rows is the probability of agreement minus the probability of disagreement, given that the column variable, \( b_{uv} \), is not 0. Somers’ \( D \) for columns is defined in a similar manner.

A discussion of all of the measures of association in this section can be found in Kendall and Stuart (1979, p. 592).
### Measures of Prediction and Uncertainty

**Optimal Prediction Coefficients:** The measures in this section do not require any ordering of the row or column variables. They are based entirely upon probabilities. Most are discussed in Bishop et al. (1975, p. 385).

Consider predicting (or classifying) the column for a given row in the table. Under the null hypothesis of independence, choose the column with the highest column marginal probability for all rows. In this case, the probability of misclassification for any row is 1 minus this marginal probability. If independence is not assumed within each row, choose the column with the highest row conditional probability. The probability of misclassification for the row becomes 1 minus this conditional probability.

Define the optimal prediction coefficient $\lambda_{c|r}$ for predicting columns from rows as the proportion of the probability of misclassification that is eliminated because the random variables are not independent. It is estimated by

$$
\lambda_{c|r} = \frac{(1 - p_{*m}) - (1 - \sum_j p_{im})}{1 - p_{*m}}
$$

where $m$ is the index of the maximum estimated probability in the row ($p_{im}$) or row margin ($p_{*m}$). A similar coefficient is defined for predicting the rows from the columns. The symmetric version of the optimal prediction $\lambda$ is obtained by summing the numerators and denominators of $\lambda_{r|c}$ and $\lambda_{c|r}$ then dividing. Standard errors for these coefficients are given in Bishop et al. (1975, p. 388).

A problem with the optimal prediction coefficients $\lambda$ is that they vary with the marginal probabilities. One way to correct this is to use row conditional probabilities. The optimal prediction $\lambda^*$ coefficients are defined as the corresponding $\lambda$ coefficients in which first the row (or column) marginals are adjusted to the same number of observations. This yields

$$
\lambda_{c|r}^* = \frac{\sum_i \max_j p_{ji} - \max_j (\sum_i p_{ji})}{R - \max_j (\sum_i p_{ji})}
$$

where $i$ indexes the rows, $j$ indexes the columns, and $p_{ji}$ is the (estimated) probability of column $j$ given row $i$.

$$
\lambda_{r|c}^*
$$

is similarly defined.

**Goodman and Kruskal $\tau$:** A second kind of prediction measure attempts to explain the proportion of the explained variation of the row (column) measure given the column (row) measure. Define the total variation in the rows as follows:

$$
n/2 - (\sum_l x_{li}^2) / (2n)
$$
Note that this is $1/(2n)$ times the sums of squares of the $a_{uv}$ variables.

With this definition of variation, the Goodman and Kruskal $\tau$ coefficient for rows is computed as the reduction of the total variation for rows accounted for by the columns, divided by the total variation for the rows. To compute the reduction in the total variation of the rows accounted for by the columns, note that the total variation for the rows within column $j$ is defined as follows:

$$q_j = x_{j*}/2 - (\sum_i x_{ij}^2)/ (2x_{**})$$

The total variation for rows within columns is the sum of the $q_j$ variables. Consistent with the usual methods in the analysis of variance, the reduction in the total variation is given as the difference between the total variation for rows and the total variation for rows within the columns.

Goodman and Kruskal’s $\tau$ for columns is similarly defined. See Bishop et al. (1975, p. 391) for the standard errors.

**Uncertainty Coefficients:** The uncertainty coefficient for rows is the increase in the log-likelihood that is achieved by the most general model over the independence model, divided by the marginal log-likelihood for the rows. This is given by the following equation:

$$U_{r|c} = \frac{\sum_{i,j} x_{ij} \log(x_{i*}x_{j*}/nx_{ij})}{\sum_i x_{i*} \log(x_{i*}/n)}$$

The uncertainty coefficient for columns is similarly defined. The symmetric uncertainty coefficient contains the same numerator as $U_{r|c}$ and $U_{c|r}$ but averages the denominators of these two statistics. Standard errors for $U$ are given in Brown (1983).

**Kruskal-Wallis:** The Kruskal-Wallis statistic for rows is a one-way analysis-of-variance-type test that assumes the column variable is monotonically ordered. It tests the null hypothesis that no row populations are identical, using average ranks for the column variable. The Kruskal-Wallis statistic for columns is similarly defined. Conover (1980) discusses the Kruskal-Wallis test.

**Test for Linear Trend:** When there are two rows, it is possible to test for a linear trend in the row probabilities if it is assumed that the column variable is monotonically ordered. In this test, the probabilities for row 1 are predicted by the column index using weighted simple linear regression. This slope is given by

$$\hat{\beta} = \frac{\sum_j x_{j*} \left( x_{1j}/x_{j*} - x_{1*}/n \right) (j - \bar{j})}{\sum_j x_{j*} (j - \bar{j})^2}$$

where
\[ j = \frac{\sum_j x_{\cdot j} j}{n} \]

is the average column index. An asymptotic test that the slope is 0 may then be obtained (in large samples) as the usual regression test of zero slope.

In two-column data, a similar test for a linear trend in the column probabilities is computed. This test assumes that the rows are monotonically ordered.

**Kappa:** Kappa is a measure of agreement computed on square tables only. In the kappa statistic, the rows and columns correspond to the responses of two judges. The judges agree along the diagonal and disagree off the diagonal. Let

\[ p_0 = \frac{\sum_i x_{ii}}{n} \]

denote the probability that the two judges agree, and let

\[ p_c = \frac{\sum_i e_{ii}}{n} \]

denote the expected probability of agreement under the independence model. Kappa is then given by

\[ \frac{(p_0 - p_c)}{(1 - p_c)}. \]

**McNemar Tests:** The McNemar test is a test of symmetry in a square contingency table. In other words, it is a test of the null hypothesis \( H_0 : \theta_{ij} = \theta_{ji} \). The multiple degrees-of-freedom version of the McNemar test with \( r (r - 1)/2 \) degrees of freedom is computed as follows:

\[ \sum_{i < j} \frac{(x_{ij} - x_{ji})^2}{(x_{ij} + x_{ji})} \]

The single degree-of-freedom test assumes that the differences, \( x_{ij} - x_{ji} \), are all in one direction. The single degree-of-freedom test will be more powerful than the multiple degrees-of-freedom test when this is the case. The test statistic is given as follows:

\[ \frac{\left( \sum_{i < j} (x_{ij} - x_{ji}) \right)^2}{\sum_{i < j} (x_{ij} + x_{ji})} \]

The exact probability can be computed by the binomial distribution.
Properties

ChiSquared
public double ChiSquared {get; }

Description
Returns the Pearson chi-squared test statistic.

Property Value
A double scalar containing the Pearson chi-squared test statistic.

ContingencyCoef
public double ContingencyCoef {get; }

Description
Returns contingency coefficient.

Property Value
A double scalar containing the contingency coefficient based on Pearson chi-squared statistic.

CramersV
public double CramersV {get; }

Description
Returns Cramer’s V.

Property Value
A double scalar containing the Cramer’s V based on Pearson chi-squared statistic.

DegreesOfFreedom
public int DegreesOfFreedom {get; }

Description
Returns the degrees of freedom for the chi-squared tests associated with the table.

Property Value
An int scalar containing the degrees of freedom for the chi-squared tests associated with the table.

ExactMean
public double ExactMean {get; }

Description
Returns the exact mean.

Property Value
A double scalar containing the exact mean based on Pearson’s chi-square statistic.

ExactStdev
public double ExactStdev {get; }
**Description**
Returns the exact standard deviation.

**Property Value**
A double scalar containing the exact standard deviation based on Pearson’s chi-square statistic.

**GSquared**
public double GSquared {get; }

**Description**
Returns the likelihood ratio $G^2$ (chi-squared).

**Property Value**
A double scalar containing the likelihood ratio $G^2$ (chi-squared).

**GSquaredP**
public double GSquaredP {get; }

**Description**
Returns the probability of a larger $G^2$ (chi-squared).

**Property Value**
A double scalar containing the probability of a larger $G^2$ (chi-squared).

**NumberOfProcessors**
public int NumberOfProcessors {get; set; }

**Description**
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**
An int indicating the maximum possible number of processors to use.

**Remarks**
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**P**
public double P {get; }

**Description**
Returns the Pearson chi-squared $p$-value for independence of rows and columns.
**Property Value**
A double scalar containing the Pearson chi-squared $p$-value for independence of rows and columns.

**Phi**

```java
public double Phi {get; }
```

**Description**
Returns phi.

**Property Value**
A double scalar containing the phi based on Pearson chi-squared statistic.

**Constructor**

**ContingencyTable**

```java
public ContingencyTable(double[,] table)
```

**Description**
Constructs and performs a chi-squared analysis of a two-way contingency table.

**Parameter**
- `table` – A double matrix containing the observed counts in the contingency table.

**Methods**

**GetContributions**

```java
public double[,] GetContributions()
```

**Description**
Returns the contributions to chi-squared for each cell in the table.

**Returns**
A double matrix of size `(table.GetLength(0) + 1) * (table.GetLength(1) + 1)` containing the contributions to chi-squared for each cell in the table.

**Remarks**
The last row and column contain the total contribution to chi-squared for that row or column.

**GetExpectedValues**

```java
public double[,] GetExpectedValues()
```

**Description**
Returns the expected values of each cell in the table.
**Returns**
A double matrix of size 
\( (\text{table.GetLength(0)} + 1) \times (\text{table.GetLength(1)} + 1) \)
containing the expected values of each cell in the table, under the null hypothesis.

**Remarks**
The marginal totals are in the last row and column.

**GetStatistics**
```csharp
public double[,] GetStatistics()
```

**Description**
Returns the statistics associated with this table.

**Returns**
A double matrix of size 23 * 5 containing statistics associated with this table.

**Remarks**
Each row corresponds to a statistic.

<table>
<thead>
<tr>
<th>Row</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>gamma</td>
</tr>
<tr>
<td>1</td>
<td>Kendall’s ( \tau )</td>
</tr>
<tr>
<td>2</td>
<td>Stuart’s ( \tau )</td>
</tr>
<tr>
<td>3</td>
<td>Somers’ D for rows (given columns)</td>
</tr>
<tr>
<td>4</td>
<td>Somers’ D for columns (given rows)</td>
</tr>
<tr>
<td>5</td>
<td>product moment correlation</td>
</tr>
<tr>
<td>6</td>
<td>Spearman rank correlation</td>
</tr>
<tr>
<td>7</td>
<td>Goodman and Kruskal ( \tau ) for rows</td>
</tr>
<tr>
<td>8</td>
<td>Goodman and Kruskal ( \tau ) for columns</td>
</tr>
<tr>
<td>9</td>
<td>uncertainty coefficient ( U ) (symmetric)</td>
</tr>
<tr>
<td>10</td>
<td>uncertainty ( U_{c</td>
</tr>
<tr>
<td>11</td>
<td>uncertainty ( U_{r</td>
</tr>
<tr>
<td>12</td>
<td>optimal prediction ( \lambda ) (symmetric)</td>
</tr>
<tr>
<td>13</td>
<td>optimal prediction ( \lambda_{r</td>
</tr>
<tr>
<td>14</td>
<td>optimal prediction ( \lambda_{c</td>
</tr>
<tr>
<td>15</td>
<td>optimal prediction ( \lambda^{*}_{r</td>
</tr>
<tr>
<td>16</td>
<td>optimal prediction ( \lambda^{*}_{c</td>
</tr>
<tr>
<td>17</td>
<td>Test for linear trend in row probabilities if ( \text{table.GetLength(0)} = 2 ). Test for linear trend in column probabilities if ( \text{table.GetLength(1)} = 2 ) and ( \text{table.GetLength(0)} ) is not 2.</td>
</tr>
<tr>
<td>18</td>
<td>Kruskal-Wallis test for no row effect</td>
</tr>
<tr>
<td>19</td>
<td>Kruskal-Wallis test for no column effect</td>
</tr>
<tr>
<td>20</td>
<td>kappa (square tables only)</td>
</tr>
<tr>
<td>21</td>
<td>McNemar test of symmetry (square tables only)</td>
</tr>
<tr>
<td>22</td>
<td>McNemar one degree of freedom test of symmetry (square tables only)</td>
</tr>
</tbody>
</table>
The columns are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>estimated statistic</td>
</tr>
<tr>
<td>1</td>
<td>standard error for any parameter value</td>
</tr>
<tr>
<td>2</td>
<td>standard error under the null hypothesis</td>
</tr>
<tr>
<td>3</td>
<td>t value for testing the null hypothesis</td>
</tr>
<tr>
<td>4</td>
<td>p-value of the test in column 3</td>
</tr>
</tbody>
</table>

If a statistic cannot be computed, or if some value is not relevant for the computed statistic, the entry is NaN (Not a Number).

In the McNemar tests, column 0 contains the statistic, column 1 contains the chi-squared degrees of freedom, column 3 contains the exact p-value (1 degree of freedom only), and column 4 contains the chi-squared asymptotic p-value. The Kruskal-Wallis test is the same except no exact p-value is computed.

**Example 1: Contingency Table**

The following example is taken from Kendall and Stuart (1979) and involves the distance vision in the right and left eyes.

```csharp
using System;
using Imsl.Stat;

class ContingencyTableEx1
{
    public static void Main(String[] args)
    {
        double[,] table = {{821, 112, 85, 35},
                           {116, 494, 145, 27},
                           {72, 151, 583, 87},
                           {43, 34, 106, 331}};
        ContingencyTable ct = new ContingencyTable(table);
        Console.Out.WriteLine("P-value = " + ct.P);
    }
}
```

**Output**

P-value = 0

**Example 2: Contingency Table**

The following example, which illustrates the use of Kappa and McNemar tests, uses the same distance vision data as in Example 1.

```csharp
using System;
using Imsl.Stat;
```
public class ContingencyTableEx2
{
    public static void Main(String[] args)
    {
        double[,] table = {{821.0, 112.0, 85.0, 35.0},
                           {116.0, 494.0, 145.0, 27.0},
                           {72.0, 151.0, 583.0, 87.0},
                           {43.0, 34.0, 106.0, 331.0}};
        String[] rlabels = new String[]{"Gamma", "Tau B",
                                         "Tau C", "D-Row",
                                         "D-Column", "Correlation",
                                         "Spearman", "GK tau rows",
                                         "GK tau cols.", "U - sym.
                                         "U - rows", "U - cols."
                                         "Lambda-sym.", "Lambda-row"
                                         "Lambda-col.", "l-star-rows"
                                         "l-star-col."
                                         "Lin. trend"
                                         "Kruskal row"
                                         "Kruskal col.", "Kappa"
                                         "McNemar",
                                         "McNemar df=1");
        ContingencyTable ct = new ContingencyTable(table);
        Console.Out.WriteLine("Pearson chi-squared statistic = " +
                             ct.ChiSquared.ToString("0.0000"));
        Console.Out.WriteLine("p-value for Pearson chi-squared = " +
                             ct.P.ToString("0.0000");
        Console.Out.WriteLine("degrees of freedom = " +
                             ct.DegreesOfFreedom);
        Console.Out.WriteLine("G-squared statistic = " +
                             ct.GSquared.ToString("0.0000");
        Console.Out.WriteLine("p-value for G-squared = " +
                             ct.GSquaredP.ToString("0.0000");
        Console.Out.WriteLine("degrees of freedom = " +
                             ct.DegreesOfFreedom);

        PrintMatrix pm = new PrintMatrix("\n  \*  \*  Table Values  \*  \*  ");
        PrintMatrixFormat pmf = new PrintMatrixFormat();
        pmf.NumberFormat = "0.00";
        pm.Print(pmf, table);
        pm.SetTitle("\n  \*  \*  Expected Values  \*  \*  ");
        pm.Print(pmf, ct.GetExpectedValues());
        pmf.NumberFormat = "0.0000";
        pm.SetTitle("\n  \*  \*  Contributions to Chi-squared  \*  \*  ");
        pm.Print(pmf, ct.GetContributions());
        Console.Out.WriteLine("\n  \*  \*  Chi-square Statistics  \*  \*  ");
        Console.Out.WriteLine("Exact mean = " + ct.ExactMean.ToString("0.0000");
Console.Out.WriteLine("Exact standard deviation = " +
    ct.ExactStdev.ToString("0.0000");
Console.Out.WriteLine("Phi = " + ct.Phi.ToString("0.0000");
Console.Out.WriteLine("P = " + ct.ContingencyCoef.ToString("0.0000");
Console.Out.WriteLine("Cramer’s V = " + ct.CramersV.ToString("0.0000");

Console.Out.WriteLine("\n    stat. std. err. " + "std. err.(Ho) t-value(Ho) p-value");
double[,] stat = ct.GetStatistics();
for (int i = 0; i < stat.GetLength(0); i++)
{
    System.Text.StringBuilder sb =
        new System.Text.StringBuilder(rlabels[i]);

    int len = sb.Length;
    for (int j = 0; j < (13 - len); j++)
        sb.Append(' ');
    sb.Append(stat[i,0].ToString("0.0000"));

    len = sb.Length;
    for (int j = 0; j < (24 - len); j++)
        sb.Append(' ');
    sb.Append(stat[i,1].ToString("0.0000"));

    len = sb.Length;
    for (int j = 0; j < (36 - len); j++)
        sb.Append(' ');
    sb.Append(stat[i,2].ToString("0.0000"));

    len = sb.Length;
    for (int j = 0; j < (50 - len); j++)
        sb.Append(' ');
    sb.Append(stat[i,3].ToString("0.0000"));

    len = sb.Length;
    for (int j = 0; j < (63 - len); j++)
        sb.Append(' ');
    sb.Append(stat[i,4].ToString("0.0000"));

    Console.Out.WriteLine(sb.ToString());
}

Output

Pearson chi-squared statistic = 3304.3684
p-value for Pearson chi-squared = 0.0000
degrees of freedom = 9
G-squared statistic = 2781.0190
p-value for G-squared = 0.0000
degrees of freedom = 9
### Table Values

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>821.00</td>
<td>112.00</td>
<td>85.00</td>
<td>35.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>116.00</td>
<td>494.00</td>
<td>145.00</td>
<td>27.00</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>72.00</td>
<td>151.00</td>
<td>583.00</td>
<td>87.00</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>43.00</td>
<td>34.00</td>
<td>106.00</td>
<td>331.00</td>
<td></td>
</tr>
</tbody>
</table>

### Expected Values

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>341.69</td>
<td>256.92</td>
<td>298.49</td>
<td>155.90</td>
<td>1053.00</td>
</tr>
<tr>
<td>1</td>
<td>253.75</td>
<td>190.80</td>
<td>221.67</td>
<td>115.78</td>
<td>782.00</td>
</tr>
<tr>
<td>2</td>
<td>289.77</td>
<td>217.88</td>
<td>253.14</td>
<td>132.21</td>
<td>893.00</td>
</tr>
<tr>
<td>3</td>
<td>166.79</td>
<td>125.41</td>
<td>145.70</td>
<td>76.10</td>
<td>514.00</td>
</tr>
<tr>
<td>4</td>
<td>1052.00</td>
<td>791.00</td>
<td>919.00</td>
<td>480.00</td>
<td>3242.00</td>
</tr>
</tbody>
</table>

### Contributions to Chi-squared

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>672.3626</td>
<td>81.7416</td>
<td>152.6959</td>
<td>93.7612</td>
<td>1000.5613</td>
</tr>
<tr>
<td>1</td>
<td>74.7802</td>
<td>481.8351</td>
<td>26.5189</td>
<td>651.2109</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>163.6605</td>
<td>20.5287</td>
<td>429.8489</td>
<td>15.4625</td>
<td>629.5006</td>
</tr>
<tr>
<td>3</td>
<td>91.8743</td>
<td>66.6263</td>
<td>10.8183</td>
<td>853.7768</td>
<td>1023.0957</td>
</tr>
<tr>
<td>4</td>
<td>1002.6776</td>
<td>650.7317</td>
<td>619.8819</td>
<td>1031.0772</td>
<td>3304.3684</td>
</tr>
</tbody>
</table>

### Chi-square Statistics

- Exact mean = 9.0028
- Exact standard deviation = 4.2402
- Phi = 1.0096
- Cramer's V = 0.5829

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
<th>Standard Error</th>
<th>Standard Error Ho</th>
<th>t-value Ho</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamma</td>
<td>0.7757</td>
<td>0.0123</td>
<td>0.0149</td>
<td>52.1897</td>
<td>0.0000</td>
</tr>
<tr>
<td>Tau B</td>
<td>0.6429</td>
<td>0.0122</td>
<td>0.0123</td>
<td>52.1897</td>
<td>0.0000</td>
</tr>
<tr>
<td>Tau C</td>
<td>0.6293</td>
<td>0.0121</td>
<td>NaN</td>
<td>52.1897</td>
<td>0.0000</td>
</tr>
<tr>
<td>D-Row</td>
<td>0.6418</td>
<td>0.0122</td>
<td>0.0123</td>
<td>52.1897</td>
<td>0.0000</td>
</tr>
<tr>
<td>D-Column</td>
<td>0.6439</td>
<td>0.0122</td>
<td>0.0123</td>
<td>52.1897</td>
<td>0.0000</td>
</tr>
<tr>
<td>Correlation</td>
<td>0.6926</td>
<td>0.0128</td>
<td>0.0172</td>
<td>40.2669</td>
<td>0.0000</td>
</tr>
<tr>
<td>Spearman</td>
<td>0.6939</td>
<td>0.0127</td>
<td>0.0127</td>
<td>54.6614</td>
<td>0.0000</td>
</tr>
<tr>
<td>GK tau rows</td>
<td>0.3420</td>
<td>0.0123</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>GK tau cols</td>
<td>0.3430</td>
<td>0.0122</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>U - sym.</td>
<td>0.3171</td>
<td>0.0110</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>U - rows</td>
<td>0.3178</td>
<td>0.0110</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>U - cols.</td>
<td>0.3164</td>
<td>0.0110</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>Lambda-sym.</td>
<td>0.5373</td>
<td>0.0124</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>Lambda-row</td>
<td>0.5374</td>
<td>0.0126</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>Lambda-col.</td>
<td>0.5372</td>
<td>0.0126</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>l-star-rows</td>
<td>0.5506</td>
<td>0.0136</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>l-star-col.</td>
<td>0.5636</td>
<td>0.0127</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>Lin. trend</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>Kruskal row</td>
<td>1561.4859</td>
<td>3.0000</td>
<td>NaN</td>
<td>NaN</td>
<td>0.0000</td>
</tr>
<tr>
<td>Kruskal col.</td>
<td>1563.0303</td>
<td>3.0000</td>
<td>NaN</td>
<td>NaN</td>
<td>0.0000</td>
</tr>
<tr>
<td>Kappa</td>
<td>0.5744</td>
<td>0.0111</td>
<td>0.0106</td>
<td>54.3583</td>
<td>0.0000</td>
</tr>
<tr>
<td>McNemar</td>
<td>4.7625</td>
<td>6.0000</td>
<td>NaN</td>
<td>NaN</td>
<td>0.5746</td>
</tr>
<tr>
<td>McNemar df=1</td>
<td>0.9487</td>
<td>1.0000</td>
<td>NaN</td>
<td>0.3459</td>
<td>0.3301</td>
</tr>
</tbody>
</table>
CategoricalGenLinModel Class

public class Imsl.Stat.CategoricalGenLinModel

Analyzes categorical data using logistic, probit, Poisson, and other linear models.

Reweighted least squares is used to compute (extended) maximum likelihood estimates in some
generalized linear models involving categorized data. One of several models, including probit, logistic,
Poisson, logarithmic, and negative binomial models, may be fit for input point or interval observations.
(In the usual case, only point observations are observed.)

Let
\[ \gamma_i = w_i + x_i^T \beta = w_i + \eta_i \]
be the linear response where \( x_i \) is a design column vector obtained from a row of \( x \), \( \beta \) is the column
vector of coefficients to be estimated, and \( w_i \) is a fixed parameter that may be input in \( x \). When some of
the \( \gamma \) are infinite at the supremum of the likelihood, then extended maximum likelihood estimates
are computed. Extended maximum likelihood is computed as the finite (but nonunique) estimates \( \hat{\beta} \) that
optimize the likelihood containing only the observations with finite \( \hat{\gamma} \). These estimates, when combined
with the set of indices of the observations such that \( \hat{\gamma} \) is infinite at the supremum of the likelihood, are
called extended maximum estimates. When none of the optimal \( \hat{\gamma} \) are infinite, extended maximum
likelihood estimates are identical to maximum likelihood estimates. Extended maximum likelihood
estimation is discussed in more detail by Clarkson and Jennrich (1991). In CategoricalGenLinModel,
observations with potentially infinite
\[ \hat{\eta}_i = x_i^T \hat{\beta} \]
are detected and removed from the likelihood if
Imsl.Stat.CategoricalGenLinModel.InfiniteEstimateMethod (p. 756) = 0. See below.

The models available in CategoricalGenLinModel are:

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Parameterization</th>
<th>Response PDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model0 (Poisson)</td>
<td>( \lambda = N \times e^{w_i + \eta} )</td>
<td>( f(y) = \lambda^y e^{-\lambda} / y! )</td>
</tr>
<tr>
<td>Model1 (Negative Binomial)</td>
<td>( \theta = \frac{e^{w_i + \eta}}{1 + e^{w_i + \eta}} )</td>
<td>( f(y) = \left( \frac{S + y - 1}{y - 1} \right) \theta^y (1 - \theta)^y )</td>
</tr>
<tr>
<td>Model2 (Logarithmic)</td>
<td>( \theta = \frac{e^{w_i + \eta}}{1 + e^{w_i + \eta}} )</td>
<td>( f(y) = \frac{(1 - \theta)^y}{y \ln \theta} )</td>
</tr>
<tr>
<td>Model3 (Logistic)</td>
<td>( \theta = \frac{e^{w_i + \eta}}{1 + e^{w_i + \eta}} )</td>
<td>( f(y) = \frac{N}{y} \theta^y (1 - \theta)^{N-y} )</td>
</tr>
<tr>
<td>Model4 (Probit)</td>
<td>( \theta = \Phi(w + \eta) )</td>
<td>( f(y) = \frac{N}{y} \theta^y (1 - \theta)^{N-y} )</td>
</tr>
<tr>
<td>Model5 (Log-log)</td>
<td>( \theta = 1 - e^{-e^{w_i + \eta}} )</td>
<td>( f(y) = \frac{N}{y} \theta^y (1 - \theta)^{N-y} )</td>
</tr>
</tbody>
</table>

Here \( \Phi \) denotes the cumulative normal distribution, \( N \) and \( S \) are known parameters specified for each
observation via column Imsl.Stat.CategoricalGenLinModel.OptionalDistributionParameterColumn (p. 759) of \( x \), and \( w \) is an optional fixed parameter specified for each observation via column.
Imsl.Stat.CategoricalGenLinModel.FixedParameterColumn (p. 755) of \(x\). (By default \(N\) is taken to be 1 for \(model = 0, 3, 4\) and 5 and \(S\) is taken to be 1 for \(model = 1\). By default \(w\) is taken to be 0.) Since the log-log model (\(model = 5\)) probabilities are not symmetric with respect to 0.5, quantitatively, as well as qualitatively, different models result when the definitions of “success” and “failure” are interchanged in this distribution. In this model and all other models involving \(\theta\), \(\theta\) is taken to be the probability of a “success.”

Note that each row vector in the data matrix can represent a single observation; or, through the use of column Imsl.Stat.CategoricalGenLinModel.FrequencyColumn (p. 755) of the matrix \(x\), each vector can represent several observations.

**Computational Details**

For interval observations, the probability of the observation is computed by summing the probability distribution function over the range of values in the observation interval. For right-interval observations, \(\Pr(Y \geq y)\) is computed as a sum based upon the equality \(\Pr(Y \geq y) = 1 - \Pr(Y < y)\). Derivatives are similarly computed. CategoricalGenLinModel allows three types of interval observations. In full interval observations, both the lower and the upper endpoints of the interval must be specified. For right-interval observations, only the lower endpoint need be given while for left-interval observations, only the upper endpoint is given.

The computations proceed as follows:

1. The input parameters are checked for consistency and validity.

2. Estimates of the means of the “independent” or design variables are computed. The frequency of the observation in all but the binomial distribution model is taken from column FrequencyColumn of the data matrix \(x\). In binomial distribution models, the frequency is taken as the product of \(n = x[i,\text{OptionalDistributionParameterColumn}]\) and \(x[i,\text{FrequencyColumn}]\). In all cases these values default to 1. Means are computed as

\[
\bar{x} = \frac{\sum_i f_i x_i}{\sum_i f_i}
\]

3. If init = 0, initial estimates of the coefficients are obtained (based upon the observation intervals) as multiple regression estimates relating transformed observation probabilities to the observation design vector. For example, in the binomial distribution models, \(\theta\) for point observations may be estimated as

\[
\hat{\theta} = x[i,\text{LowerEndpointColumn}] / x[i,\text{OptionalDistributionParameterColumn}]
\]

and, when \(model = 3\), the linear relationship is given by

\[
(\ln(\hat{\theta} / (1 - \hat{\theta})) \approx x\beta)
\]

while if \(model = 4\),

\[
(\Phi^{-1}(\hat{\theta}) = x\beta)
\]

For bounded interval observations, the midpoint of the interval is used for \(x[i,\text{Imsl.Stat.CategoricalGenLinModel.LowerEndpointColumn}]\) (p. 757).
Right-interval observations are not used in obtaining initial estimates when the distribution has unbounded support (since the midpoint of the interval is not defined). When computing initial estimates, standard modifications are made to prevent illegal operations such as division by zero. Regression estimates are obtained at this point, as well as later, by use of linear regression.

4. Newton-Raphson iteration for the maximum likelihood estimates is implemented via iteratively reweighted least squares. Let

\[ \Psi(x^T_i \beta) \]

denote the log of the probability of the \(i\)-th observation for coefficients \(\beta\). In the least-squares model, the weight of the \(i\)-th observation is taken as the absolute value of the second derivative of

\[ \Psi(x^T_i \beta) \]

with respect to

\[ \gamma_i = x^T_i \beta \]

(times the frequency of the observation), and the dependent variable is taken as the first derivative \(\Psi\) with respect to \(\gamma_i\), divided by the square root of the weight times the frequency. The Newton step is given by

\[ \Delta \beta = \left( \sum_i |\Psi''(\gamma_i)x_i x_i^T|^{-1} \right)^{-1} \sum_i \Psi'(\gamma_i)x_i \]

where all derivatives are evaluated at the current estimate of \(\gamma\), and \(\beta_{n+1} = \beta_n - \Delta \beta\). This step is computed as the estimated regression coefficients in the least-squares model. Step halving is used when necessary to ensure a decrease in the criterion.

5. Convergence is assumed when the maximum relative change in any coefficient update from one iteration to the next is less than Imsl.Stat.CategoricalGenLinModel.ConvergenceTolerance (p. 753) or when the relative change in the log-likelihood from one iteration to the next is less than \(\text{ConvergenceTolerance}/100\). Convergence is also assumed after Imsl.Stat.CategoricalGenLinModel.MaxIterations (p. 758) or when step halving leads to a step size of less than .0001 with no increase in the log-likelihood.

6. For interval observations, the contribution to the log-likelihood is the log of the sum of the probabilities of each possible outcome in the interval. Because the distributions are discrete, the sum may involve many terms. The user should be aware that data with wide intervals can lead to expensive (in terms of computer time) computations.

7. If InfiniteEstimateMethod is set to 0, then the methods of Clarkson and Jennrich (1991) are used to check for the existence of infinite estimates in

\[ \eta_i = x_i^T \beta \]

As an example of a situation in which infinite estimates can occur, suppose that observation \(j\) is right censored with \(t_j > 15\) in a logistic model. If design matrix \(x\) is such that \(x_{jm} = 1\) and \(x_{im} = 0\) for all \(i \neq j\), then the optimal estimate of \(\beta_m\) occurs at

\[ \hat{\beta}_m = \infty \]
leading to an infinite estimate of both $\beta_m$ and $\eta_j$. In CategoricalGenLinModel, such estimates may be “computed.” In all models fit by CategoricalGenLinModel, infinite estimates can only occur when the optimal estimated probability associated with the left- or right-censored observation is 1. If InfiniteEstimateMethod is set to 0, left- or right-censored observations that have estimated probability greater than 0.995 at some point during the iterations are excluded from the log-likelihood, and the iterations proceed with a log-likelihood based upon the remaining observations. This allows convergence of the algorithm when the maximum relative change in the estimated coefficients is small and also allows for the determination of observations with infinite

$$\eta_i = x_i^T \beta$$

At convergence, linear programming is used to ensure that the eliminated observations have infinite $\eta_i$. If some (or all) of the removed observations should not have been removed (because their estimated $\eta_i$'s must be finite), then the iterations are restarted with a log-likelihood based upon the finite $\eta_i$ observations. See Clarkson and Jennrich (1991) for more details.

When InfiniteEstimateMethod is set to 1, no observations are eliminated during the iterations. In this case, when infinite estimates occur, some (or all) of the coefficient estimates $\hat{\beta}$ will become large, and it is likely that the Hessian will become (numerically) singular prior to convergence.

When infinite estimates for the $\hat{\eta}_i$ are detected, linear regression (see Chapter 2, Regression:) is used at the convergence of the algorithm to obtain unique estimates $\hat{\beta}$. This is accomplished by regressing the optimal $\hat{\eta}_i$ or the observations with finite $\eta$ against $x\beta$, yielding a unique $\hat{\beta}$ (by setting coefficients $\hat{\beta}$ that are linearly related to previous coefficients in the model to zero). All of the final statistics relating to $\hat{\beta}$ are based upon these estimates.

8. Residuals are computed according to methods discussed by Pregibon (1981). Let $\ell_i(\gamma_i)$ denote the log-likelihood of the $i$-th observation evaluated at $\gamma_i$. Then, the standardized residual is computed as

$$r_i = \frac{\ell'_i(\hat{\gamma}_i)}{\sqrt{\ell''_i(\hat{\gamma}_i)}}$$

where $\hat{\gamma}_i$ is the value of $\gamma_i$ when evaluated at the optimal $\hat{\beta}$ and the derivatives here (and only here) are with respect to $\gamma$ rather than with respect to $\beta$. The denominator of this expression is used as the “standard error of the residual” while the numerator is the “raw” residual.

Following Cook and Weisberg (1982), we take the influence of the $i$-th observation to be

$$\ell'_i(\hat{\gamma}_i)^T \ell''(\hat{\gamma}_i)^{-1} \ell(\hat{\gamma}_i)$$

This quantity is a one-step approximation to the change in the estimates when the $i$-th observation is deleted. Here, the partial derivatives are with respect to $\beta$.

**Programming Notes**

1. Classification variables are specified via
   
   `Imsl.Stat.CategoricalGenLinModel.ClassificationVariableColumn` (p. 752). Indicator or dummy variables are created for the classification variables.
2. To enhance precision “centering” of covariates is performed if
Imsl.Stat.CategoricalGenLinModel.ModelIntercept (p. 758) is set to 1 and (number of
observations) - (number of rows in x missing one or more values) > 1. In doing so, the sample
means of the design variables are subtracted from each observation prior to its inclusion in the
model. On convergence the intercept, its variance and its covariance with the remaining estimates
are transformed to the uncentered estimate values.

3. Two methods for specifying a binomial distribution model are possible. In the first method,
x[i,FrequencyColumn] contains the frequency of the observation while
x[i,LowerEndpointColumn] is 0 or 1 depending upon whether the observation is a success or
failure. In this case, \( N = x[i,OptionalDistributionParameterColumn] \) is always 1. The
model is treated as repeated Bernoulli trials, and interval observations are not possible.

A second method for specifying binomial models is to use \( x[i,LowerEndpointColumn] \) to represent
the number of successes in the \( x[i,OptionalDistributionParameterColumn] \) trials. In this case,
\( x[i,FrequencyColumn] \) will usually be 1, but it may be greater than 1, in which case interval
observations are possible.

Note that the Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) method must be called before using any
property as a right operand, otherwise the value is null.

### Properties

**CaseAnalysis**

virtual public double[,] CaseAnalysis {get; }

**Description**
The case analysis.

**Property Value**
A double matrix containing the case analysis or null if Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) has not been called.

**Remarks**
The matrix is \( nobs \times 5 \) where \( nobs \) is the number of observations. The matrix contains:

<table>
<thead>
<tr>
<th>Column</th>
<th>Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Prediction.</td>
</tr>
<tr>
<td>1</td>
<td>The residual.</td>
</tr>
<tr>
<td>2</td>
<td>The estimated standard error of the residual.</td>
</tr>
<tr>
<td>3</td>
<td>The estimated influence of the observation.</td>
</tr>
<tr>
<td>4</td>
<td>The standardized residual.</td>
</tr>
</tbody>
</table>

Case studies are computed for all observations except where missing values prevent their computation. The prediction in column 0 depends upon the model used as follows:
<table>
<thead>
<tr>
<th>Model</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The predicted mean for the observation.</td>
</tr>
<tr>
<td>1-4</td>
<td>The probability of a success on a single trial.</td>
</tr>
</tbody>
</table>

**CensorColumn**

virtual public int CensorColumn {set; }

**Description**
The column number in x which contains the interval type for each observation.

**Property Value**
An int scalar which indicates the column number x which contains the interval type code for each observation.

**Remarks**
The valid codes are interpreted as:

<table>
<thead>
<tr>
<th>x[i,CensorColumn]</th>
<th>Censoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Point observation. The response is unique and is given by x[i,LowerEndpointColumn].</td>
</tr>
<tr>
<td>1</td>
<td>Right interval. The response is greater than or equal to x[i,LowerEndpointColumn] and less than or equal to the upper bound, if any, of the distribution.</td>
</tr>
<tr>
<td>2</td>
<td>Left interval. The response is less than or equal to x[i,UpperEndpointColumn] and greater than or equal to the lower bound of the distribution.</td>
</tr>
<tr>
<td>3</td>
<td>Full interval. The response is greater than or equal to x[i,LowerEndpointColumn] but less than or equal to x[i,UpperEndpointColumn].</td>
</tr>
</tbody>
</table>

By default, CensorColumn = 0.

**Exception**

System.ArgumentException is thrown when CensorColumn is less than 0 or greater than or equal to the number of columns of x

**ClassificationVariableColumn**

virtual public int[] ClassificationVariableColumn {set; }

**Description**
An index vector to contain the column numbers in x that are classification variables.

**Property Value**
An int vector which contains the column numbers in x that are classification variables.

**Remarks**
By default, ClassificationVariableColumn is not referenced.
Exception

System.ArgumentException is thrown when an element of ClassificationVariableColumn is less than 0 or greater than or equal to the number of columns of x.

ClassificationVariableCounts
virtual public int[] ClassificationVariableCounts {get; }

Description
The number of values taken by each classification variable.

Property Value
An int array of length nclvar containing the number of values taken by each classification variable where nclvar is the number of classification variables or null if Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) has not been called.

Exception
Imsl.Stat.ClassificationVariableException is thrown when the number of values taken by each classification variable has been set by the user to be less than or equal to 1.

ClassificationVariableValues
virtual public double[] ClassificationVariableValues {get; }

Description
The distinct values of the classification variables in ascending order.

Property Value
A double array of length \( \sum_{i=0}^{nclvar} nclval[i] \) containing the distinct values of the classification variables in ascending order where nclvar is the number of classification variables and nclval[i] is the number of values taken by the i-th classification variable.

Remarks
A null is returned if Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) has not been called prior to calling this method.

Exception
Imsl.Stat.ClassificationVariableException is thrown when the number of values taken by each classification variable has been set by the user to be less than or equal to 1.

ConvergenceTolerance
virtual public double ConvergenceTolerance {set; }

Description
The convergence criterion.

Property Value
A double scalar specifying the convergence criterion.
Remarks
Convergence is assumed when the maximum relative change in any coefficient estimate is less than ConvergenceTolerance from one iteration to the next or when the relative change in the log-likelihood, Imsl.Stat.CategoricalGenLinModel.OptimizedCriterion (p. 759), from one iteration to the next is less than ConvergenceTolerance/100. ConvergenceTolerance must be greater than 0.

By default, ConvergenceTolerance = .001.

Exception
System.ArgumentException is thrown if ConvergenceTolerance is an equal to 0.

CovarianceMatrix
virtual public double[,] CovarianceMatrix {get; }

Description
The estimated asymptotic covariance matrix of the coefficients.

Property Value
A double matrix containing the estimated asymptotic covariance matrix of the coefficients or null if Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) has not been called.

Remarks
The covariance matrix is nCoef by nCoef where nCoef is the number of coefficients in the model.

DesignVariableMeans
virtual public double[] DesignVariableMeans {get; }

Description
The means of the design variables.

Property Value
A double array of length nCoef containing the means of the design variables where nCoef is the number of coefficients in the model or null if Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) has not been called.

ExtendedLikelihoodObservations
virtual public int[] ExtendedLikelihoodObservations {get; set; }

Description
A vector indicating which observations are included in the extended likelihood.

Property Value
An int array of length nobS indicating which observations are included in the extended likelihood where nobS is the number of observations.
Remarks

ExtendedLikelihoodObservations is an int array of length nobs indicating which observations are included in the extended likelihood where nobs is the number of observations. The values within the array are interpreted as:

<table>
<thead>
<tr>
<th>Value</th>
<th>Status of observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Observation i is in the likelihood.</td>
</tr>
<tr>
<td>1</td>
<td>Observation i cannot be in the likelihood because it contains at least one missing value in x.</td>
</tr>
<tr>
<td>2</td>
<td>Observation i is not in the likelihood. Its estimated parameter is infinite.</td>
</tr>
</tbody>
</table>

A null is returned if Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) has not been called prior to calling this method.

By default, all elements are zero.

Exception

System.ArgumentException is thrown when an element of ExtendedLikelihoodObservations is not in the range [0,2]

FixedParameterColumn

virtual public int FixedParameterColumn {set; }

Description

The column number in x that contains a fixed parameter for each observation that is added to the linear response prior to computing the model parameter.

Property Value

An int scalar which indicates the column number in x that contains a fixed parameter for each observation that is added to the linear response prior to computing the model parameter.

Remarks

The “fixed” parameter allows one to test hypothesis about the parameters via the log-likelihoods. By default the fixed parameter is assumed to be zero.

Exception

System.ArgumentException is thrown when FixedParameterColumn is less than 0 or greater than or equal to the number of columns of x

FrequencyColumn

virtual public int FrequencyColumn {set; }

Description

The column number in x that contains the frequency of response for each observation.

Property Value

An int scalar which indicates the column number in x that contains the frequency of response for each observation.
Remarks
By default a frequency of 1 for each observation is assumed.

Exception
System.ArgumentException is thrown when FrequencyColumn is less than 0 or greater than or equal to the number of columns of x

Hessian
virtual public double[,] Hessian {get; }

Description
The Hessian computed at the initial parameter estimates.

Property Value
A double matrix containing the Hessian computed at the input parameter estimates.

Remarks
The Hessian matrix is \( n_{Coef} \times n_{Coef} \) where \( n_{Coef} \) is the number of coefficients in the model. This member function will call Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) to get the Hessian if the Hessian has not already been computed.

Exceptions
Imsl.Stat.ClassificationVariableException is thrown when the number of values taken by each classification variable has been set by the user to be less than or equal to 1
Imsl.Stat.ClassificationVariableLimitException is thrown when the sum of the number of distinct values taken on by each classification variable exceeds the maximum allowed, Imsl.Stat.CategoricalGenLinModel.UpperBound (p. 761)
Imsl.Stat.DeleteObservationsException is thrown if the number of observations to be deleted has grown too large
Imsl.Stat.RankDeficientException is thrown if the model has been determined to be rank deficient

InfiniteEstimateMethod
virtual public int InfiniteEstimateMethod {set; }

Description
Specifies the method used for handling infinite estimates.

Property Value
An int scalar indicating which method to use for handling infinite estimates.

Remarks
The value of InfiniteEstimateMethod is interpreted as follows:
<table>
<thead>
<tr>
<th>InfiniteEstimateMethod</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Remove a right or left-censored observation from the log-likelihood whenever the probability of the observation exceeds 0.995. At convergence, use linear programming to check that all removed observations actually have an estimated linear response that is infinite. Set ExtendedLikelihoodObservations[i] for observation i to 2 if the linear response is infinite. If not all removed observations have infinite linear response, recompute the estimates based upon the observations with estimated linear response that is finite. This option is valid only for censoring codes 1 and 2.</td>
</tr>
<tr>
<td>1</td>
<td>Iterate without checking for infinite estimates.</td>
</tr>
</tbody>
</table>

By default InfiniteEstimateMethod = 1.

Exception

System.ArgumentException is thrown when InfiniteEstimateMethod is less than 0 or greater than 1.

---

### LastParameterUpdates

```csharp
virtual public double[] LastParameterUpdates {get; }
```

**Description**

The last parameter updates (excluding step halvings).

**Property Value**

A double array of length nCoef containing the last parameter updates (excluding step halvings) or null if Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) has not been called.

---

### LowerEndpointColumn

```csharp
virtual public int LowerEndpointColumn {set; }
```

**Description**

The column number in x that contains the lower endpoint of the observation interval for full interval and right interval observations.

**Property Value**

An int scalar which indicates the column number in x that contains the lower endpoint of the observation interval for full interval and right interval observations.

**Remarks**

By default all observations are treated as “point” observations and x[i,LowerEndpointColumn] contains the observation point. If this member function is not called, the last column of x is assumed to contain the “point” observations.
Exception
System.ArgumentException is thrown when LowerEndpointColumn is less than 0 or greater than or equal to the number of columns of x

MaxIterations
virtual public int MaxIterations {set; }

Description
The maximum number of iterations allowed.

Property Value
An int specifying the maximum number of iterations allowed.

Remarks
By default, MaxIterations = 30.

Exception
System.ArgumentException is thrown if MaxIterations is less than or equal to 0

ModelIntercept
virtual public int ModelIntercept {set; }

Description
The intercept option.

Property Value
An int scalar which indicates whether or not the model has an intercept.

Remarks
Input ModelIntercept is interpreted as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No intercept is in the model (unless otherwise provided for by the user).</td>
</tr>
<tr>
<td>1</td>
<td>Intercept is automatically included in the model.</td>
</tr>
</tbody>
</table>

By default ModelIntercept = 1.

Exception
System.ArgumentException is thrown when ModelIntercept is less than 0 or greater than 1

NRowsMissing
virtual public int NRowsMissing {get; }

Description
The number of rows of data in x that contain missing values in one or more specific columns of x.

Property Value
An int scalar representing the number of rows of data in x that contain missing values in one or more specific columns of x or null if Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) has not been called.
Remarks
The columns of $x$ included in the count are the columns containing the upper or lower endpoints of full interval, left interval, or right interval observations. Also included are the columns containing the frequency responses, fixed parameters, optional distribution parameters, and interval type for each observation. Columns containing classification variables and columns associated with each effect in the model are also included.

**ObservationMax**
```csharp
virtual public int ObservationMax {set; }
```

**Description**
The maximum number of observations that can be handled in the linear programming.

**Property Value**
An `int` scalar which sets the maximum number of observations that can be handled in the linear programming.

**Remarks**
By default, ObservationMax is set to the number of observations.

**Exception**
`System.ArgumentException` is thrown if ObservationMax is less than 0.

**OptimizedCriterion**
```csharp
virtual public double OptimizedCriterion {get; }
```

**Description**
The optimized criterion.

**Property Value**
A `double` scalar representing the optimized criterion or null if `Imsl.Stat.CategoricalGenLinModel.Solve` (p. 764) has not been called.

**Remarks**
The criterion to be maximized is a constant plus the log-likelihood.

**OptionalDistributionParameterColumn**
```csharp
virtual public int OptionalDistributionParameterColumn {set; }
```

**Description**
The column number in $x$ that contains an optional distribution parameter for each observation.

**Property Value**
An `int` scalar which indicates the column number in $x$ that contains an optional distribution parameter for each observation.
Remarks

The distribution parameter values are interpreted as follows depending on the model chosen:

<table>
<thead>
<tr>
<th>Model</th>
<th>Meaning of $x[i,OptionalDistributionParameterColumn]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Poisson parameter is given by $x[i,OptionalDistributionParameterColumn] \times e^\rho$.</td>
</tr>
<tr>
<td>1</td>
<td>The number of successes required in the negative binomial is given by $x[i,OptionalDistributionParameterColumn]$.</td>
</tr>
<tr>
<td>2</td>
<td>$x[i,OptionalDistributionParameterColumn]$ is not used.</td>
</tr>
<tr>
<td>3-5</td>
<td>The number of trials in the binomial distribution is given by $x[i,OptionalDistributionParameterColumn]$.</td>
</tr>
</tbody>
</table>

By default, the distribution parameter is assumed to be 1.

Exception

[System.ArgumentException] is thrown when [OptionalDistributionParameterColumn] is less than 0 or greater than or equal to the number of columns of $x$.

Parameters

virtual public double[,] Parameters {get; }

Description

Parameter estimates and associated statistics.

Property Value

An $nCoef$ row by 4 column double matrix containing the parameter estimates and associated statistics or null if Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) has not been called.

Remarks

Here, $nCoef$ is the number of coefficients in the model. The statistics returned are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Coefficient estimate.</td>
</tr>
<tr>
<td>1</td>
<td>Estimated standard deviation of the estimated coefficient.</td>
</tr>
<tr>
<td>2</td>
<td>Asymptotic normal score for testing that the coefficient is zero.</td>
</tr>
<tr>
<td>3</td>
<td>$\rho$ - value associated with the normal score in column 2.</td>
</tr>
</tbody>
</table>

Product

virtual public double[] Product {get; }

Description

The inverse of the Hessian times the gradient vector computed at the input parameter estimates.

Property Value

A double array of length $nCoef$ containing the inverse of the Hessian times the gradient vector computed at the input parameter estimates.
Remarks

*nCoef* is the number of coefficients in the model. This member function will call Imsl.Stat.CategoricalGenLinModel.Solve (p. 764) to get the product if the product has not already been computed.

Exceptions

Imsl.Stat.ClassificationVariableException is thrown when the number of values taken by each classification variable has been set by the user to be less than or equal to 1

Imsl.Stat.ClassificationVariableLimitException is thrown when the sum of the number of distinct values taken on by each classification variable exceeds the maximum allowed, Imsl.Stat.CategoricalGenLinModel.UpperBound (p. 761)

Imsl.Stat.DeleteObservationsException is thrown if the number of observations to be deleted has grown too large

Imsl.Stat.RankDeficientException is thrown if the model has been determined to be rank deficient

Tolerance

```csharp
virtual public double Tolerance {get; set; }
```

Description

The tolerance used in determining linear dependence.

Property Value

A double value used in determining linear dependence.

Default: Tolerance = 0.22204460492503130808e-14.

Remarks

When linear dependence is detected, a RankDeficientException is thrown and no results are computed. Computations for a rank deficient model can be forced to continue by specifying a negative tolerance. If Tolerance is negative, the absolute value of Tolerance will be used to determine linear dependence, but computations will proceed with a rank deficient warning. In this case the results should be carefully inspected and used with caution.

UpperBound

```csharp
virtual public int UpperBound {set; }
```

Description

Defines the upper bound on the sum of the number of distinct values taken on by each classification variable.

Property Value

An int scalar specifying the upper bound on the sum of the number of distinct values taken on by each classification variable.

Remarks

By default, if property ClassificationVariableColumn has not been referenced UpperBound = 1. If property ClassificationVariableColumn has been referenced UpperBound = x.GetLength(0) * nclvar where nclvar is the number of classification variables.
Exception

System.ArgumentException is thrown when UpperBound is less than 1 and the number of classification variables is greater than 0.

UpperEndpointColumn

virtual public int UpperEndpointColumn {set; }

Description

The column number in x that contains the upper endpoint of the observation interval for full interval and left interval observations.

Property Value

An int scalar which indicates the column number in x that contains the upper endpoint of the observation interval for full interval and left interval observations.

Remarks

By default all observations are treated as “point” observations.

Exception

System.ArgumentException is thrown when UpperEndpointColumn is less than 0 or greater than or equal to the number of columns of x.

Constructor

CategoricalGenLinModel

public CategoricalGenLinModel(double[,] x,

Description

Constructs a new CategoricalGenLinModel.

Parameters

x – A double input matrix containing the data where the number of rows in the matrix is equal to the number of observations.
model – An int scalar which specifies the distribution of the response variable and the function used to model the distribution parameter.

Remarks

Use one of the class members from the following table. The lower bound given in the table is the minimum possible value of the response variable:
Let $\gamma$ be the dot product of a row in the design matrix with the parameters (plus the fixed parameter, if used). Then, the functions used to model the distribution parameter are given by:

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponential</td>
<td>$e^\gamma$</td>
</tr>
<tr>
<td>Logistic</td>
<td>$e^\gamma / (1 + e^\gamma)$</td>
</tr>
<tr>
<td>Probit</td>
<td>$\Phi(\gamma)$ (where $\Phi$ is the normal cdf)</td>
</tr>
<tr>
<td>Log-log</td>
<td>$1 - e^{-\gamma}$</td>
</tr>
</tbody>
</table>

### Methods

**SetEffects**

virtual public void SetEffects(int[] indef, int[] nvef)

**Description**

Initializes an index vector to contain the column numbers in $x$ associated with each effect.

**Parameters**

- `indef` – An int vector of length $\sum_{k=0}^{n_{ef}-1} n_{vef}[k]$ where $n_{ef}$ is the number of effects in the model.
- `nvef` – An int vector of length $n_{ef}$ where $n_{ef}$ is the number of effects in the model.

**Remarks**

`indef` contains the column numbers in $x$ that are associated with each effect. Member function `SetEffects(int [], nvef [])` sets the number of variables associated with each effect in the model. The first $n_{vef}[0]$ elements of `indef` give the column numbers of the variables in the first effect. The next $n_{vef}[0]$ elements give the column numbers of the variables in the second effect, etc. By default this vector is not referenced.

`nvef` contains the number of variables associated with each effect in the model. By default this vector is not referenced.

**Exception**

`System.ArgumentException` is thrown when an element of `indef` is less than 0 or greater than or equal to the number of columns of $x$ or if an element of `nvef` is less than or equal to 0

**SetInitialEstimates**

virtual public void SetInitialEstimates(int init, double[] estimates)
Description
Sets the initial parameter estimates option.

Parameters

  \( \text{init} \) – An input \( \text{int} \) indicating the desired initialization method for the initial estimates of the parameters.

  \( \text{estimates} \) – An input \( \text{double} \) array of length \( nCoef \) containing the initial estimates of the parameters where \( nCoef \) is the number of estimated coefficients in the model.

Remarks
If this method is not called, \( \text{init} \) is set to 0.

\[
\begin{array}{|c|c|}
\hline
\text{init} & \text{Action} \\
\hline
0 & \text{Unweighted linear regression is used to obtain initial estimates.} \\
1 & \text{The } nCoef, \text{ number of coefficients, elements of estimates contain initial estimates of the parameters. Use of this option requires that the user know } nCoef \text{ beforehand.} \\
\hline
\end{array}
\]

\( \text{estimates} \) is used if \( \text{init} = 1 \). If this member function is not called, unweighted linear regression is used to obtain the initial estimates.

Exception

  System.ArgumentException is thrown when \( \text{init} \) is not in the range \([0,1]\)

Solve

virtual public double[,] Solve()

Description
Returns the parameter estimates and associated statistics for a CategoricalGenLinModel object.

Returns
An \( nCoef \) row by 4 column double matrix containing the parameter estimates and associated statistics.

Remarks
Here, \( nCoef \) is the number of coefficients in the model. The statistics returned are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Coefficient estimate.</td>
</tr>
<tr>
<td>1</td>
<td>Estimated standard deviation of the estimated coefficient.</td>
</tr>
<tr>
<td>2</td>
<td>Asymptotic normal score for testing that the coefficient is zero.</td>
</tr>
<tr>
<td>3</td>
<td>( \rho ) - value associated with the normal score in column 2.</td>
</tr>
</tbody>
</table>
Exceptions

Imsl.Stat.ClassificationVariableException is thrown when the number of values taken by each classification variable has been set by the user to be less than or equal to 1.

Imsl.Stat.ClassificationVariableLimitException is thrown when the sum of the number of distinct values taken on by each classification variable exceeds the maximum allowed, Imsl.Stat.CategoricalGenLinModel.UpperBound (p. 761).

Imsl.Stat.DeleteObservationsException is thrown if the number of observations to delete has grown too large.

Imsl.Stat.RankDeficientException is thrown if the model has been determined to be rank deficient.

Example 1: Example: Mortality of beetles.

The first example is from Prentice (1976) and involves the mortality of beetles after exposure to various concentrations of carbon disulphide. Both a logit and a probit fit are produced for linear model $\mu + \beta x$.

The data is given as

<table>
<thead>
<tr>
<th>Covariate(x)</th>
<th>N</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.690</td>
<td>59</td>
<td>6</td>
</tr>
<tr>
<td>1.724</td>
<td>60</td>
<td>13</td>
</tr>
<tr>
<td>1.755</td>
<td>62</td>
<td>18</td>
</tr>
<tr>
<td>1.784</td>
<td>56</td>
<td>28</td>
</tr>
<tr>
<td>1.811</td>
<td>63</td>
<td>52</td>
</tr>
<tr>
<td>1.836</td>
<td>59</td>
<td>53</td>
</tr>
<tr>
<td>1.861</td>
<td>62</td>
<td>61</td>
</tr>
<tr>
<td>1.883</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

using System;
using Imsl.Math;
using Imsl.Stat;

public class CategoricalGenLinModelEx1
{
    public static void Main(String[] args)
    {
        // Set up a PrintMatrix object for later use.
        PrintMatrixFormat mf;
        PrintMatrix p;
        p = new PrintMatrix();
        mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();
        mf.NumberFormat = "0.0000";
        double[,] x = {

Categorical and Discrete Data Analysis  CategoricalGenLinModel • 765
CategoricalGenLinModel CATGLM3, CATGLM4;
// MODEL3
CATGLM3 = new CategoricalGenLinModel(x,
    CategoricalGenLinModel.DistributionParameterModel.Model3);
CATGLM3.LowerEndpointColumn = 2;
CATGLM3.OptionalDistributionParameterColumn = 1;
CATGLM3.InfiniteEstimateMethod = 1;
CATGLM3.ModelIntercept = 1;
int[] nvef = new int[]{1};
int[] indef = new int[]{0};
CATGLM3.SetEffects(indef, nvef);
CATGLM3.UpperBound = 1;
Console.Out.WriteLine("MODEL3");
p.SetTitle("Coefficient Statistics");
p.Print(mf, CATGLM3.Solve());
Console.Out.WriteLine("Log likelihood " + CATGLM3.OptimizedCriterion);
p.SetTitle("Asymptotic Coefficient Covariance");
p.SetMatrixType(PrintMatrix.MatrixType.UpperTriangular);
p.Print(mf, CATGLM3.CovarianceMatrix);
p.SetMatrixType(PrintMatrix.MatrixType.Full);
p.SetTitle("Case Analysis");
p.Print(mf, CATGLM3.CaseAnalysis);
p.SetTitle("Last Coefficient Update");
p.Print(CATGLM3.LastParameterUpdates);
p.SetTitle("Covariate Means");
p.Print(CATGLM3.DesignVariableMeans);
p.SetTitle("Observation Codes");
p.Print(CATGLM3.ExtendedLikelihoodObservations);
Console.Out.WriteLine("Number of Missing Values " + CATGLM3.NRowsMissing);

// MODEL4
CATGLM4 = new CategoricalGenLinModel(x,
    CategoricalGenLinModel.DistributionParameterModel.Model4);
CATGLM4.LowerEndpointColumn = 2;
CATGLM4.OptionalDistributionParameterColumn = 1;
CATGLM4.InfiniteEstimateMethod = 1;
CATGLM4.ModelIntercept = 1;
CATGLM4.SetEffects(indef, nvef);
CATGLM4.UpperBound = 1;
CATGLM4.Solve();
Console.Out.WriteLine("nMODEL4");
Console.Out.WriteLine("Log likelihood " + CATGLM4.OptimizedCriterion);
p.SetTitle("Coefficient Statistics");
p.Print(mf, CATGLM4.Parameters);
Output

MODEL3

Coefficient Statistics

\[
\begin{array}{cccc}
-60.7568 & 5.1876 & -11.7118 & 0.0000 \\
34.2985 & 2.9164 & 11.7607 & 0.0000 \\
\end{array}
\]

Log likelihood -18.7781790423339

Asymptotic Coefficient Covariance

\[
\begin{array}{cccc}
26.9117 & -15.1243 & 8.5052 \\
\end{array}
\]

Case Analysis

\[
\begin{array}{cccc}
0.0577 & 2.5934 & 1.7916 & 0.2674 & 1.4475 \\
0.1644 & 3.1390 & 2.8706 & 0.3470 & 1.0935 \\
0.3629 & -4.4976 & 3.7860 & 0.3108 & -1.1879 \\
0.6063 & -5.9517 & 3.6562 & 0.2322 & -1.6279 \\
0.7954 & 1.8901 & 3.2020 & 0.2688 & 0.5903 \\
0.9016 & -0.1949 & 2.2878 & 0.2380 & -0.0852 \\
0.9558 & 1.7434 & 1.6193 & 0.1976 & 1.0767 \\
0.9787 & 1.2783 & 1.1185 & 0.1382 & 1.1429 \\
\end{array}
\]

Last Coefficient Update

\[
\begin{array}{cccc}
0 & 1.85192237936898E-07 \\
1 & 1.33163785440457E-05 \\
\end{array}
\]

Covariate Means

\[
\begin{array}{cccc}
0 & 1.793 \\
1 & 0 \\
\end{array}
\]

Observation Codes

\[
\begin{array}{cccc}
0 & 0 \\
1 & 0 \\
2 & 0 \\
3 & 0 \\
4 & 0 \\
5 & 0 \\
6 & 0 \\
7 & 0 \\
\end{array}
\]

Number of Missing Values 0

MODEL4

Log likelihood -18.2323545743846

Coefficient Statistics
Example 2: Example: Poisson Model.

In this example, the following data illustrate the Poisson model when all types of interval data are present. The example also illustrates the use of classification variables and the detection of potentially infinite estimates (which turn out here to be finite). These potential estimates lead to the two iteration summaries. The input data is

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A linear model $\mu + \beta_1 x_1 + \beta_2 x_2$ is fit where $x_1 = 1$ if the Class 1 variable is 0, $x_1 = 1$, otherwise, and the $x_2$ variable is similarly defined.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;

public class CategoricalGenLinModelEx2
{
    public static void Main(String[] args)
    {
        // Set up a PrintMatrix object for later use.
        PrintMatrixFormat mf;
        PrintMatrix p;
        p = new PrintMatrix();
        mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();
        mf.NumberFormat = "0.0000";

        double[,] x = {
            {0.0, 5.0, 0.0, 1.0, 0.0},
            {9.0, 4.0, 3.0, 0.0, 0.0},
            {9.0, 4.0, 1.0, 0.0, 0.0},
            {9.0, 0.0, 2.0, 1.0, 1.0},
            {0.0, 1.0, 0.0, 0.0, 1.0}};

        CategoricalGenLinModel CATGLM;
        CATGLM = new CategoricalGenLinModel(x,
            CategoricalGenLinModel.DistributionParameterModel.Model0);
        CATGLM.UpperEndpointColumn = 0;
        CATGLM.LowerEndpointColumn = 1;
        CATGLM.OptionalDistributionParameterColumn = 1;
    }
}
```
CATGLM.CensorColumn = 2;
CATGLM.InfiniteEstimateMethod = 0;
CATGLM.ModelIntercept = 1;
int[] indcl = new int[]{3, 4};
CATGLM.ClassificationVariableColumn = indcl;
int[] nvef = new int[]{1, 1};
int[] indef = new int[]{3, 4};
CATGLM.SetEffects(indef, nvef);
CATGLM.UpperBound = 4;

p.SetTitle("Coefficient Statistics");
p.Print(mf, CATGLM.Solve());
Console.Out.WriteLine("Log likelihood "+ CATGLM.OptimizedCriterion);
p.SetTitle("Asymptotic Coefficient Covariance");
p.SetMatrixType(PrintMatrix.MatrixType.UpperTriangular);
p.Print(mf, CATGLM.CovarianceMatrix);
p.SetMatrixType(PrintMatrix.MatrixType.Full);
p.SetTitle("Case Analysis");
p.Print(mf, CATGLM.CaseAnalysis);
p.SetTitle("Last Coefficient Update");
p.Print(CATGLM.LastParameterUpdates);
p.SetTitle("Covariate Means");
p.Print(CATGLM.DesignVariableMeans);
p.SetTitle("Distinct Values For Each Class Variable");
p.Print(CATGLM.ClassificationVariableValues);
Console.Out.WriteLine("Number of Missing Values "+ CATGLM.NRowsMissing);

Output

Coefficient Statistics

-0.5488  1.1713  -0.4685  0.6395
0.5488  0.6098  0.8999  0.3684
0.5488  1.0825  0.5069  0.6123

Log likelihood -3.11463849257844

Asymptotic Coefficient Covariance

1.3719  -0.3719  -1.1719
0.3719   0.1719
1.1719

Case Analysis

5.0000  0.0000  2.2361  1.0000  0.0000
6.9246  -0.4122  2.1078  0.7636  -0.1955
6.9246  0.4122  1.1727  0.2364  0.3515
0.0000  0.0000  0.0000  0.0000  NaN
1.0000  0.0000  1.0000  1.0000  0.0000

Last Coefficient Update

0
CategoricalGenLinModel.DistributionParameterModel

Enumeration


Indicates the function used to model the distribution parameter.

Fields

Model0
public Imsl.Stat.CategoricalGenLinModel.DistributionParameterModel Model0

Description
Indicates an exponential function is used to model the distribution parameter. The distribution of the response variable is Poisson. The lower bound of the response variable is 0.

Model1
public Imsl.Stat.CategoricalGenLinModel.DistributionParameterModel Model1

Description
Indicates a logistic function is used to model the distribution parameter. The distribution of the response variable is negative Binomial. The lower bound of the response variable is 0.

Model2
Description
Indicates a logistic function is used to model the distribution parameter. The distribution of the response variable is Logarithmic. The lower bound of the response variable is 1.

Model3

Description
Indicates a logistic function is used to model the distribution parameter. The distribution of the response variable is Binomial. The lower bound of the response variable is 0.

Model4
public Imsl.Stat.CategoricalGenLinModel.DistributionParameterModel Model4

Description
Indicates a probit function is used to model the distribution parameter. The distribution of the response variable is Binomial. The lower bound of the response variable is 0.

Model5

Description
Indicates a log-log function is used to model the distribution parameter. The distribution of the response variable is Binomial. The lower bound of the response variable is 0.
Chapter 16: Nonparametric Statistics

Types

class SignTest ................................................................. 773
class WilcoxonRankSum .................................................. 777

Usage Notes

Much of what is considered nonparametric statistics is included in other chapters. Topics of possible interest in other chapters are: nonparametric measures of location and scale (see “Basic Statistics”), nonparametric measures in a contingency table (see “Categorical and Discrete Data Analysis”), measures of correlation in a contingency table (see “Correlation and Covariance”), and tests of goodness of fit and randomness (see “Tests of Goodness of Fit and Randomness”).

Missing Values

Most classes described in this chapter automatically handle missing values (NaN, “Not a Number”; see Double.NaN).

Tied Observations

The WilcoxonRankSum class described in this chapter contains a set method, setFuzz. Observations that are within fuzz of each other in absolute value are said to be tied. If fuzz = 0.0, observations must be identically equal before they are considered to be tied. Other positive values of fuzz allow for numerical imprecision or roundoff error.

SignTest Class

public class Imsl.Stat.SignTest
Performs a sign test.

Class `SignTest` tests hypotheses about the proportion $p$ of a population that lies below a value $q$, where $p$ and $q$ correspond to the `Percentage` and `Percentile` properties, respectively. In continuous distributions, this can be a test that $q$ is the 100 $p$-th percentile of the population from which $x$ was obtained. To carry out testing, `SignTest` tallies the number of values above $q$ in the number of positive differences $x[j-1] - \text{Percentile}$ for $j = 1, 2, \ldots, \text{x.length}$. The binomial probability of the number of values above $q$ in the number of positive differences $x[j-1] - \text{Percentile}$ for $j = 1, 2, \ldots, \text{x.length}$ or more values above $q$ is then computed using the proportion $p$ and the sample size in $x$ (adjusted for the missing observations and ties).

Hypothesis testing is performed as follows for the usual null and alternative hypotheses:

- $H_0 : Pr(x \leq q) \geq p$ (the $p$-th quantile is at least $q$)
  
  $H_1 : Pr(x \leq q) < p$
  
  Reject $H_0$ if probability is less than or equal to the significance level.

- $H_0 : Pr(x \leq q) \leq p$ (the $p$-th quantile is at least $q$)
  
  $H_1 : Pr(x \leq q) > p$
  
  Reject $H_0$ if probability is greater than or equal to 1 minus the significance level.

- $H_0 : Pr(x = q) = p$ (the $p$-th quantile is $q$)
  
  $H_1 : Pr((x \leq q) < p) \text{ or } Pr((x \leq q) > p)$
  
  Reject $H_0$ if probability is less than or equal to half the significance level or greater than or equal to 1 minus half the significance level.

The assumptions are as follows:

1. They are independent and identically distributed.
2. Measurement scale is at least ordinal; i.e., an ordering less than, greater than, and equal to exists in the observations.

Many uses for the sign test are possible with various values of $p$ and $q$. For example, to perform a matched sample test that the difference of the medians of $y$ and $z$ is 0.0, let $p = 0.5$, $q = 0.0$, and $x_i = y_i - z_i$ in matched observations $y$ and $z$. To test that the median difference is $c$, let $q = c$.

## Properties

### NumPositiveDev

```csharp
public int NumPositiveDev { get; }
```

**Description**

Returns the number of positive differences. Note that the `Compute` method must be invoked first before invoking this method. Otherwise, the return value is 0.
Property Value
A scalar int containing the number of positive differences $x[j-1] - \text{Percentile}$ for $j = 1, 2, ..., x.\text{Length}$.

**NumZeroDev**
public int NumZeroDev {get; }  

**Description**
Returns the number of zero differences. Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is 0.

**Property Value**
A scalar int containing the number of zero differences (ties) $x[j-1] - \text{Percentile}$ for $j = 1, 2, ..., x.\text{Length}$.

**Percentage**
public double Percentage {get; set; }  

**Description**
The percentage percentile of the population.

**Property Value**
A double scalar containing the value in the range (0, 1).

**Remarks**
$\text{Percentile}$ is the $100 \times \text{percentage}$ percentile of the population. By default, $\text{Percentage} = 0.5$.

**Percentile**
public double Percentile {get; set; }  

**Description**
The hypothesized percentile of the population.

**Property Value**
A double scalar containing the hypothesized percentile of the population from which $x$ was drawn.

**Remarks**
By default, $\text{Percentile} = 0.0$.

**Constructor**

**SignTest**
public SignTest(double[] x)
Description
Constructor for SignTest.

Parameter
   x – A double array containing the data.

Method

Compute
public double Compute()

Description
Performs a sign test.

Returns
A double scalar containing the Binomial probability of NumPositiveDev or more positive differences in x.length - number of zero differences trials.

Remarks
Call this value probability. If using default values, the null hypothesis is that the median equals 0.0.

Example 1: Sign Test
This example tests the hypothesis that at least 50 percent of a population is negative. Because 0.18 < 0.95, the null hypothesis at the 5-percent level of significance is not rejected.

```csharp
using System;
using Imsl.Stat;

public class SignTestEx1
{
    public static void Main(String[] args)
    {
        double[] x = new double[]{ 92.0, 139.0, - 6.0,
                                   10.0, 81.0, - 11.0,
                                   45.0, - 25.0, - 4.0,
                                   22.0, 2.0, 41.0,
                                   13.0, 8.0, 33.0,
                                   45.0, - 33.0, - 45.0,
                                   - 12.0};

        SignTest st = new SignTest(x);

        Console.Out.WriteLine
             ("Probability = " + st.Compute().ToString("0.000000");
    }
}
```
Output
Probability = 0.179642

Example 2: Sign Test
This example tests the null hypothesis that at least 75 percent of a population is negative. Because 0.923 < 0.95, the null hypothesis at the 5-percent level of significance is rejected.

```csharp
using System;
using Imsl.Stat;

public class SignTestEx2
{
    public static void Main(String[] args)
    {
        double[] x = new double[] { 92.0, 139.0, -6.0, 10.0, 81.0, -11.0, 45.0, -25.0, -4.0, 22.0, 2.0, 41.0, 13.0, 8.0, 33.0, 45.0, -33.0, -45.0, -12.0};

        SignTest st = new SignTest(x);
        st.Percentage = 0.75;
        st.Percentile = 0.0;
        Console.Out.WriteLine("Probability = " + st.Compute().ToString("0.000000");
        Console.Out.WriteLine("Number of positive deviations = " + st.NumPositiveDev);
        Console.Out.WriteLine("Number of ties = " + st.NumZeroDev);
    }
}
```

Output
Probability = 0.922543
Number of positive deviations = 12
Number of ties = 0

WilcoxonRankSum Class

```csharp
public class Imsl.Stat.WilcoxonRankSum

performs a Wilcoxon rank sum test.

Class WilcoxonRankSum performs the Wilcoxon rank sum test for identical population distribution functions. The Wilcoxon test is a linear transformation of the Mann-Whitney U test. If the difference
between the two populations can be attributed solely to a difference in location, then the Wilcoxon test
becomes a test of equality of the population means (or medians) and is the nonparametric equivalent of
the two-sample \(t\)-test. Class \texttt{WilcoxonRankSum} obtains ranks in the combined sample after first
eliminating missing values from the data. The rank sum statistic is then computed as the sum of the
ranks in the \(x\) sample. Three methods for handling ties are used. (A tie is counted when two observations
are within \(\varepsilon\)uzz of each other.) Method 1 uses the largest possible rank for tied observations in the
smallest sample, while Method 2 uses the smallest possible rank for these observations. Thus, the range
of possible rank sums is obtained.

Method 3 for handling tied observations between samples uses the average rank of the tied observations.
Asymptotic standard normal scores are computed for the \(W\) score (based on a variance that has been
adjusted for ties) when average ranks are used (see Conover 1980, p. 217), and the probability associated
with the two-sided alternative is computed.

**Hypothesis Tests**

In each of the following tests, the first line gives the hypothesis (and its alternative) under the
assumptions 1 to 3 below, while the second line gives the hypothesis when assumption 4 is also true. The
rejection region is the same for both hypotheses and is given in terms of Method 3 for handling ties. If
another method for handling ties is desired, another output statistic, \texttt{stat[0]} or \texttt{stat[3]}, should be
used, where \texttt{stat} is the array containing the statistics returned from the \texttt{getStatistics} method.

<table>
<thead>
<tr>
<th>Test</th>
<th>Null Hypothesis</th>
<th>Alternative Hypothesis</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(H_0: \text{Pr}(x_1 &lt; x_2) = 0.5) (H_0: E(x_1) = E(x_2))</td>
<td>(H_1: \text{Pr}(x_1 &lt; x_2) \neq 0.5) (H_1: E(x_1) \neq E(x_2))</td>
<td>Reject if \texttt{stat[9]} is less than the significance level of the test. Alternatively, reject the null hypothesis if \texttt{stat[6]} is too large or too small.</td>
</tr>
<tr>
<td>2</td>
<td>(H_0: \text{Pr}(x_1 &lt; x_2) \leq 0.5) (H_0: E(x_1) \geq E(x_2))</td>
<td>(H_1: \text{Pr}(x_1 &lt; x_2) \neq 0.5) (H_1: E(x_1) &lt; E(x_2))</td>
<td>Reject if \texttt{stat[6]} is too small</td>
</tr>
<tr>
<td>3</td>
<td>(H_0: \text{Pr}(x_1 &lt; x_2) \geq 0.5) (H_0: E(x_1) \leq E(x_2))</td>
<td>(H_1: \text{Pr}(x_1 &lt; x_2) &lt; 0.5) (H_1: E(x_1) &gt; E(x_2))</td>
<td>Reject if \texttt{stat[6]} is too large</td>
</tr>
</tbody>
</table>

**Assumptions**

1. \(x\) and \(y\) contain random samples from their respective populations.
2. All observations are mutually independent.
3. The measurement scale is at least ordinal (i.e., an ordering less than, greater than, or equal to
exists among the observations).
4. If \(f(x)\) and \(g(y)\) are the distribution functions of \(x\) and \(y\), then \(g(y) = f(x + c)\) for some constant
\(c\) (i.e., the distribution of \(y\) is, at worst, a translation of the distribution of \(x\)).

Tables of critical values of the \(W\) statistic are given in the references for small samples.

778 • \texttt{WilcoxonRankSum}  
IMSL C# Numerical Library
Constructor

WilcoxonRankSum

public WilcoxonRankSum(double[] x, double[] y)

Description

Constructor for WilcoxonRankSum.

Parameters

- x – A double array containing the first sample.
- y – A double array containing the second sample.

Methods

Compute

public double Compute()

Description

Performs a Wilcoxon rank sum test.

Returns

A double scalar containing the two-sided p-value for the Wilcoxon rank sum statistic that is computed with average ranks used in the case of ties.

GetStatistics

public double[] GetStatistics()

Description

Returns the statistics. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns

A double array of length 10 containing statistics.

Remarks

The statistics are as follows:
<table>
<thead>
<tr>
<th>Row</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Wilcoxon W statistic (the sum of the ranks of the x observations) adjusted for ties in such a manner that W is as small as possible</td>
</tr>
<tr>
<td>1</td>
<td>2 x E(W) - W, where E(W) is the expected value of W</td>
</tr>
<tr>
<td>2</td>
<td>probability of obtaining a statistic less than or equal to ( \min{W, 2 \times E(W) - W} )</td>
</tr>
<tr>
<td>3</td>
<td>W statistic adjusted for ties in such a manner that W is as large as possible</td>
</tr>
<tr>
<td>4</td>
<td>2 x E(W) - W, where E(W) is the expected value of W, adjusted for ties in such a manner that W is as large as possible</td>
</tr>
<tr>
<td>5</td>
<td>probability of obtaining a statistic less than or equal to ( \min{W, 2 \times E(W) - W} ), adjusted for ties in such a manner that W is as large as possible</td>
</tr>
<tr>
<td>6</td>
<td>W statistic with average ranks used in case of ties</td>
</tr>
<tr>
<td>7</td>
<td>estimated standard error of Row 6 under the null hypothesis of no difference</td>
</tr>
<tr>
<td>8</td>
<td>standard normal score associated with Row 6</td>
</tr>
<tr>
<td>9</td>
<td>two-sided p-value associated with Row 8</td>
</tr>
</tbody>
</table>

**SetFuzz**

public void SetFuzz(double fuzz)

**Description**

Sets the nonnegative constant used to determine ties in computing ranks in the combined samples.

**Parameter**

- **fuzz** – A double scalar containing the nonnegative constant used to determine ties in computing ranks in the combined samples.

**Remarks**

A tie is declared when two observations in the combined sample are within **fuzz** of each other. By default, **fuzz** = \( 100 \times 2.2204460492503131e - 16 \times \max(|x_1|, |x_2|) \).

**Example 1: Wilcoxon Rank Sum Test**

The following example is taken from Conover (1980, p. 224). It involves the mixing time of two mixing machines using a total of 10 batches of a certain kind of batter, five batches for each machine. The null hypothesis is not rejected at the 5-percent level of significance.

```csharp
using System;
using Imsl;
using Imsl.Stat;

public class WilcoxonRankSumEx1
{
    public static void Main(String[] args)
    {
```

---

**SetFuzz**

public void SetFuzz(double fuzz)

**Description**

Sets the nonnegative constant used to determine ties in computing ranks in the combined samples.

**Parameter**

- **fuzz** – A double scalar containing the nonnegative constant used to determine ties in computing ranks in the combined samples.

**Remarks**

A tie is declared when two observations in the combined sample are within **fuzz** of each other. By default, **fuzz** = \( 100 \times 2.2204460492503131e - 16 \times \max(|x_1|, |x_2|) \).

**Example 1: Wilcoxon Rank Sum Test**

The following example is taken from Conover (1980, p. 224). It involves the mixing time of two mixing machines using a total of 10 batches of a certain kind of batter, five batches for each machine. The null hypothesis is not rejected at the 5-percent level of significance.

```csharp
using System;
using Imsl;
using Imsl.Stat;

public class WilcoxonRankSumEx1
{
    public static void Main(String[] args)
    {
```
double[] x = new double[]{7.3, 6.9, 7.2, 7.8, 7.2};
double[] y = new double[]{7.4, 6.8, 6.9, 6.7, 7.1};

WilcoxonRankSum wilcoxon = new WilcoxonRankSum(x, y);
Console.Out.WriteLine
("p-value = " + wilcoxon.Compute().ToString("0.0000");
}
}
Output

Wilcoxon W statistic ...................... 34.000
2*E(W) - W ................................ 21.000
p-value ................................... 0.110
Adjusted Wilcoxon statistic ............... 35.000
Adjusted 2*E(W) - W ....................... 20.000
Adjusted p-value .......................... 0.075
W statistics for averaged ranks............ 34.500
Standard error of W (averaged ranks) ...... 4.758
Standard normal score of W (averaged ranks) 1.471
Two-sided p-value of W (averaged ranks) ... 0.141

Both sample sizes, "x.Length" and "y.Length", are less than 25.
Significance levels should be obtained from tabled values.
Imsl.Stat.WilcoxonRankSum: At least one tie is detected between the samples.
Chapter 17: Tests of Goodness of Fit

Types

class ChiSquaredTest ................................................................. 783
class NormalityTest ................................................................. 788
class KolmogorovOneSample ....................................................... 793
class KolmogorovTwoSample ...................................................... 796

Usage Notes

The classes in this chapter are used to test for goodness of fit. The goodness-of-fit tests are described in Conover (1980). There is a goodness-of-fit test for general distributions and a chi-squared test. The user supplies the hypothesized cumulative distribution function for the test. There is a class that can be used to test specifically for the normal distribution.

The chi-squared goodness-of-fit test may be used with discrete as well as continuous distributions. The chi-squared goodness-of-fit test allows for missing values (NaN, not a number) in the input data.

The Kolmogorov-Smirnov routines in this chapter compute exact probabilities in small to moderate sample sizes. The chi-squared goodness-of-fit test may be used with discrete as well as continuous distributions.

The Kolmogorov-Smirnov and chi-squared goodness-of-fit test routines allow for missing values (NaN, not a number) in the input data. The routines that test for randomness do not allow for missing values.

ChiSquaredTest Class

public class Imsl.Stat.ChiSquaredTest
Chi-squared goodness-of-fit test.
ChiSquaredTest performs a chi-squared goodness-of-fit test that a random sample of observations is distributed according to a specified theoretical cumulative distribution. The theoretical distribution, which may be continuous, discrete, or a mixture of discrete and continuous distributions, is specified via a user-defined function \( F \) where \( F \) implements ICdfFunction. Because the user is allowed to specify a range for the observations in the SetRange method, a test that is conditional upon the specified range is performed.

ChiSquaredTest can be constructed in two different ways. The intervals can be specified via the array cutpoints. Otherwise, the number of cutpoints can be given and equiprobable intervals computed by the constructor. The observations are divided into these intervals. Regardless of the method used to obtain them, the intervals are such that the lower endpoint is not included in the interval while the upper endpoint is always included. The user should determine the cutpoints when the cumulative distribution function has discrete elements since ChiSquaredTest cannot determine them in this case.

By default, the lower and upper endpoints of the first and last intervals are \(-\infty\) and \(+\infty\), respectively. The method SetRange can be used to change the range.

A tally of counts is maintained for the observations in \( x \) as follows:

If the cutpoints are specified by the user, the tally is made in the interval to which \( x_i \) belongs, using the user-specified endpoints.

If the cutpoints are determined by the class then the cumulative probability at \( x_i, F(x_i) \), is computed using Cdf.

The tally for \( x_i \) is made in interval number \( \lfloor mf(x) + 1 \rfloor \), where \( m \) is the number of categories and \( \lfloor . \rfloor \) is the function that takes the greatest integer that is no larger than the argument of the function. If the cutpoints are specified by the user, the tally is made in the interval to which \( x_i \) belongs using the endpoints specified by the user. Thus, if the computer time required to calculate the cumulative distribution function is large, user-specified cutpoints may be preferred in order to reduce the total computing time.

If the expected count in any cell is less than 1, then a rule of thumb is that the chi-squared approximation may be suspect. A warning message to this effect is issued in this case, as well as when an expected value is less than 5.

### Properties

**ChiSquared**

```csharp
public double ChiSquared {get; }
```

**Description**
The chi-squared statistic.

**Property Value**
A double containing the chi-squared statistic.

**DegreesOfFreedom**

```csharp
public double DegreesOfFreedom {get; }
```
Description
Returns the degrees of freedom in chi-squared.

Property Value
A double containing the degrees of freedom in the chi-squared statistic.

\[ P \]
\[ \text{public double } P \{ \get; } \]

Description
The \( p \)-value for the chi-squared statistic.

Property Value
A double containing the \( p \)-value for the chi-squared statistic.

Constructors

\textbf{ChiSquaredTest}
\[ \text{public ChiSquaredTest(Imsl.Stat.ICdfFunction } cdf, \text{ double[]} \text{ cutpoints, int } \text{nParameters}) \]

Description
Constructor for the Chi-squared goodness-of-fit test.

Parameters
\begin{itemize}
  \item \( cdf \) – Object that implements the ICdfFunction interface.
  \item \( \text{cutpoints} \) – A double array containing the cutpoints.
  \item \( \text{nParameters} \) – A int which specifies the number of parameters estimated in computing the Cdf.
\end{itemize}

Exception
\begin{itemize}
  \item \text{Imsl.Stat.NotCDFException} is thrown if the function \( cdf.CdfFunction \) is not a valid CDF.
\end{itemize}

\textbf{ChiSquaredTest}
\[ \text{public ChiSquaredTest(Imsl.Stat.ICdfFunction } cdf, \text{ int } \text{nCutpoints, int } \text{nParameters}) \]

Description
Constructor for the Chi-squared goodness-of-fit test

Parameters
\begin{itemize}
  \item \( cdf \) – Object that implements the ICdfFunction interface.
  \item \( \text{nCutpoints} \) – A int which specifies the number of cutpoints.
  \item \( \text{nParameters} \) – A int which specifies the number of parameters estimated in computing the Cdf.
\end{itemize}
Exceptions

- `Imsl.Stat.NotCDFException` is thrown if the function `cdf.CdfFunction` is not a valid CDF.
- `Imsl.Stat.DidNotConvergeException` is thrown if the iteration to find the inverse of the CDF did not converge. The inverse CDF is needed to compute the cutpoints.

Methods

**GetCellCounts**

```csharp
public double[] GetCellCounts()
```

**Description**

Returns the cell counts.

**Returns**

A `double` array which contains the number of actual observations in each cell.

**GetCutpoints**

```csharp
public double[] GetCutpoints()
```

**Description**

Returns the cutpoints.

**Returns**

A `double` array which contains the cutpoints.

**Remarks**

The intervals defined by the cutpoints are such that the lower endpoint is not included while the upper endpoint is included in the interval.

**GetExpectedCounts**

```csharp
public double[] GetExpectedCounts()
```

**Description**

Returns the expected counts.

**Returns**

A `double` array which contains the number of expected observations in each cell.

**SetCutpoints**

```csharp
public void SetCutpoints(double[] cutpoints)
```

**Description**

Sets the cutpoints.

**Parameter**

- `cutpoints` – A `double` array which contains the cutpoints.
Remarks
The intervals defined by the cutpoints are such that the lower endpoint is not included while the upper endpoint is included in the interval.

SetRange
public void SetRange(double lower, double upper)

Description
Sets endpoints of the range of the distribution.

Parameters
   lower – A double which specifies the lower range limit.
   upper – A double which specifies the upper range limit.

Remarks
Points outside of the range are ignored so that distributions conditional on the range can be used. In this case, the point lower is excluded from the first interval, but the point upper is included in the last interval.
By default, a range on the whole real line is used.

Update
public void Update(double[] x, double[] freq)

Description
Adds new observations to the test.

Parameters
   x – A double array which contains the new observations to be added to the test.
   freq – A double array which contains the frequencies of the corresponding new observations in x.

Update
public void Update(double x, double freq)

Description
Adds a new observation to the test.

Parameters
   x – A double which specifies the new observation to be added to the test.
   freq – A double which specifies the frequency of the new observation, x.

Example: The Chi-squared Goodness-of-fit Test
In this example, a discrete binomial random sample of size 1000 with binomial parameter $p = 0.3$ and binomial sample size 5 is generated via Random.nextBinomial. Random.setSeed is first used to set the seed. After the ChiSquaredTest constructor is called, the random observations are added to the test one at a time to simulate streaming data. The Chi-squared statistic, p-value, and Degrees of freedom are then computed and printed.
using System;
using Imsl.Stat;

public class ChiSquaredTestEx1 : ICdfFunction
{
    public double CdfFunction(double x)
    {
        return Cdf.Binomial((int) x, 5, 0.3);
    }

    public static void Main(String[] args)
    {
        // Seed the random number generator
        rn.Multiplier = 16807;

        // Construct a ChiSquaredTest object
        ICdfFunction bindf = new ChiSquaredTestEx1();
        double[] cutp = new double[]{0.5, 1.5, 2.5, 3.5, 4.5};
        int nParameters = 0;
        ChiSquaredTest cst =
            new ChiSquaredTest(bindf, cutp, nParameters);
        for (int i = 0; i < 1000; i++)
        {
            cst.Update(rn.NextBinomial(5, 0.3), 1.0);
        }

        // Print goodness-of-fit test statistics
        Console.Out.WriteLine("The Chi-squared statistic is " + cst.ChiSquared);
        Console.Out.WriteLine("The P-value is " + cst.P);
        Console.Out.WriteLine("The Degrees of freedom are " + cst.DegreesOfFreedom);
    }
}

Output
The Chi-squared statistic is 4.79629666357389
The P-value is 0.441242957205526
The Degrees of freedom are 5
Imsl.Stat.ChiSquaredTest: An expected value is less than five.

---

NormalityTest Class

public class Imsl.Stat.NormalityTest

Performs a test for normality.
Three methods are provided for testing normality: the Shapiro-Wilk W test, the Lilliefors test, and the chi-squared test.

**Shapiro-Wilk W Test**

The Shapiro-Wilk W test is thought by D’Agostino and Stevens (1986, p. 406) to be one of the best omnibus tests of normality. The function is based on the approximations and code given by Royston (1982a, b, c). It can be used in samples as large as 2,000 or as small as 3. In the Shapiro and Wilk test, $W$ is given by

$$W = \frac{\left( \sum a_i x_{(i)} \right)^2}{\left( \sum (x_i - \bar{x})^2 \right)}$$

where $x_{(i)}$ is the $i$-th largest order statistic and $x$ is the sample mean. Royston (1982) gives approximations and tabled values that can be used to compute the coefficients $a_i, i = 1, \ldots, n$, and obtains the significance level of the $W$ statistic.

**Lilliefors Test**

This function computes Lilliefors test and its $p$-values for a normal distribution in which both the mean and variance are estimated. The one-sample, two-sided Kolmogorov-Smirnov statistic $D$ is first computed. The $p$-values are then computed using an analytic approximation given by Dallal and Wilkinson (1986). Because Dallal and Wilkinson give approximations in the range (0.01, 0.10) if the computed probability of a greater $D$ is less than 0.01, the $p$-value is set to 0.50. Note that because parameters are estimated, $p$-values in Lilliefors test are not the same as in the Kolmogorov-Smirnov Test.

Observations should not be tied. If tied observations are found, an informational message is printed. A general reference for the Lilliefors test is Conover (1980). The original reference for the test for normality is Lilliefors (1967).

**Chi-Squared Test**

This function computes the chi-squared statistic, its $p$-value, and the degrees of freedom of the test. Argument $n$ finds the number of intervals into which the observations are to be divided. The intervals are equiprobable except for the first and last interval, which are infinite in length.

If more flexibility is desired for the specification of intervals, the same test can be performed with class ChiSquaredTest.

**Properties**

**ChiSquared**

`public double ChiSquared {get; }`

**Description**

Returns the chi-square statistic for the chi-squared goodness-of-fit test.
**Property Value**
A double scalar containing the chi-square statistic.

**Remarks**
Returns Double.NaN for other tests.

**DegreesOfFreedom**
```csharp
public double DegreesOfFreedom {get; }
```

**Description**
Returns the degrees of freedom for the chi-squared goodness-of-fit test.

**Property Value**
A double scalar containing the degrees of freedom.

**Remarks**
Returns Double.NaN for other tests.

**MaxDifference**
```csharp
public double MaxDifference {get; }
```

**Description**
Returns the maximum absolute difference between the empirical and the theoretical distributions for the Lilliefors test.

**Property Value**
A double scalar containing the maximum absolute difference between the empirical and the theoretical distributions.

**Remarks**
Returns Double.NaN for other tests.

**ShapiroWilkW**
```csharp
public double ShapiroWilk {get; }
```

**Description**
Returns the Shapiro-Wilk W statistic for the Shapiro-Wilk W test.

**Property Value**
A double scalar containing the Shapiro-Wilk W statistic.

**Remarks**
Returns Double.NaN for other tests.

---

**Constructor**

**NormalityTest**
```csharp
public NormalityTest(double[] x)
```
Description
Constructor for NormalityTest.

Parameter
x – A double array containing the observations.

Remarks
x.length must be in the range from 3 to 2,000, inclusive, for the Shapiro-Wilk W test and must be greater than 4 for the Lilliefors test.

Methods

ChiSquaredTest
public double ChiSquaredTest(int n)

Description
Performs the chi-squared goodness-of-fit test.

Parameter
n – A int scalar containing the number of cells into which the observations are to be tallied.

Returns
A double scalar containing the p-value for the chi-squared goodness-of-fit test.

Exceptions
Imsl.Stat.NoVariationInputException is thrown if there is no variation in the input data
Imsl.Stat.DidNotConvergeException is thrown if the iteration did not converge


LillieforsTest
public double LillieforsTest()

Description
Performs the Lilliefors test.

Returns
A double scalar containing the p-value for the Lilliefors test.

Remarks
Probabilities less than 0.01 are reported as 0.01, and probabilities greater than 0.10 for the normal distribution are reported as 0.5. Otherwise, an approximate probability is computed.

Tests of Goodness of Fit
NormalityTest • 791
Exceptions

Imsl.Stat.NoVariationInputException is thrown if there is no variation in the input data
Imsl.Stat.DidNotConvergeException is thrown if the iteration did not converge

ShapiroWilkTest

public double ShapiroWilkTest()

Description

Performs the Shapiro-Wilk W test using Royston’s calculation.

Returns

A double scalar containing the p-value for the Shapiro-Wilk W test.

Exceptions

Imsl.Stat.NoVariationInputException is thrown if there is no variation in the input data
Imsl.Stat.DidNotConvergeException is thrown if the iteration did not converge

ShapiroWilkWTest

public double ShapiroWilkWTest()

Description

Performs the Shapiro-Wilk W test.

Returns

A double scalar containing the p-value for the Shapiro-Wilk W test.

Exceptions

Imsl.Stat.NoVariationInputException is thrown if there is no variation in the input data
Imsl.Stat.DidNotConvergeException is thrown if the iteration did not converge

Example: Shapiro-Wilk W Test

The following example is taken from Conover (1980, pp. 195, 364). The data consists of 50 two-digit numbers taken from a telephone book. The W test fails to reject the null hypothesis of normality at the .05 level of significance.

using System;
using Imsl;
using Imsl.Stat;

public class NormalityTestEx1
{
    public static void Main(String[] args)
    {
        double[] x = new double[]
        {
            23.0, 36.0, 54.0, 61.0, 73.0, 23.0,
            37.0, 54.0, 61.0, 73.0, 24.0, 40.0,
            56.0, 62.0, 74.0, 27.0, 42.0, 57.0,
            63.0, 75.0, 29.0, 43.0, 57.0, 64.0,
        };
    }
}
```csharp
NormalityTest nt = new NormalityTest(x);
Console.Out.WriteLine
("p-value = " + nt.ShapiroWilkWTest().ToString("0.0000");
Console.Out.WriteLine("Shapiro Wilk W Statistic = "+
nt.ShapiroWilkW.ToString("0.0000");
```

Output

p-value = 0.2309
Shapiro Wilk W Statistic = 0.9642

### KolmogorovOneSample Class

```csharp
public class Imsl.Stat.KolmogorovOneSample

The class `KolmogorovOneSample` performs a Kolmogorov-Smirnov goodness-of-fit test in one sample.

The hypotheses tested follow:

\[
H_0 : F(x) = F^*(x) \quad H_1 : F(x) \neq F^*(x) \\
H_0 : F(x) \geq F^*(x) \quad H_1 : F(x) < F^*(x) \\
H_0 : F(x) \leq F^*(x) \quad H_1 : F(x) > F^*(x)
\]

where \( F \) is the cumulative distribution function (CDF) of the random variable, and the theoretical cdf, \( F^* \), is specified via the user-supplied function cdf. Let \( n \) be the number of observations minus the number of missing observations. The test statistics for both one-sided alternatives \( D^+_n \) and \( D^-_n \) and the two-sided \( D_n \) alternative are computed as well as an asymptotic z-score and \( p \)-values associated with the one-sided and two-sided hypotheses. For \( n > 80 \), asymptotic \( p \)-values are used (see Gibbons 1971). For \( n \leq 80 \), exact one-sided \( p \)-values are computed according to a method given by Conover (1980, page 350). An approximate two-sided test \( p \)-value is obtained as twice the one-sided \( p \)-value. The approximation is very close for one-sided \( p \)-values less than 0.10 and becomes very bad as the one-sided \( p \)-values get larger.

The theoretical CDF is assumed to be continuous. If the CDF is not continuous, the statistics \( D^*_n \) will not be computed correctly.

Estimation of parameters in the theoretical CDF from the sample data will tend to make the \( p \)-values associated with the test statistics too liberal. The empirical CDF will tend to be closer to the theoretical CDF than it should be.

Tests of Goodness of Fit

KolmogorovOneSample
No attempt is made to check that all points in the sample are in the support of the theoretical CDF. If all sample points are not in the support of the CDF, the null hypothesis must be rejected.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaximumDifference</td>
<td>$D^+$, the maximum difference between the theoretical and empirical CDF's.</td>
</tr>
<tr>
<td>MinimumDifference</td>
<td>$D^-$, the minimum difference between the theoretical and empirical CDF's.</td>
</tr>
<tr>
<td>NumberMissing</td>
<td>The number of missing values in the data.</td>
</tr>
<tr>
<td>NumberOfTies</td>
<td>The number of ties in the data.</td>
</tr>
<tr>
<td>OneSidedPValue</td>
<td>Probability of the statistic exceeding $D$ under the null hypothesis of equality and against the one-sided alternative. An exact probability is computed if the number of observation is less than or equal to 80, otherwise an approximate probability is computed.</td>
</tr>
<tr>
<td>TestStatistic</td>
<td>The test statistic, $D = \max(D^+, D^-)$.</td>
</tr>
<tr>
<td>TwoSidedPValue</td>
<td></td>
</tr>
</tbody>
</table>
**Description**

Probability of the statistic exceeding $D$ under the null hypothesis of equality and against the two-sided alternative.

**Remarks**

This probability is twice the probability, $p_1$, reported by OneSidedPValue, (or 1.0 if $p_1 \geq 1/2$). This approximation is nearly exact when $p_1 < 0.1$.

**Z**

```java
public double Z {get; }
```

**Description**

The normalized $D$ statistic without the continuity correction applied.

---

**Constructor**

**KolmogorovOneSample**

```java
public KolmogorovOneSample(Imsl.Stat.ICdfFunction cdf, double[] x)
```

**Description**

Constructs a one sample Kolmogorov-Smirnov goodness-of-fit test.

**Parameters**

- `cdf` – is the cdf function, $F(x)$. If must be non-decreasing and its value must be in [0, 1].
- `x` – is a double array containing the observations.

---

**Example: Kolmogorov One Sample**

In this example, a random sample of size 100 is generated using class Random for the uniform (0, 1) distribution. We want to test the null hypothesis that the cdf is the standard normal distribution with a mean of 0.5 and a variance equal to the uniform (0, 1) variance (1/12).

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;

public class KolmogorovOneSampleEx1 : ICdfFunction
{
    public static void Main(String[] args)
    {
        double[] x = new double[100];
        random.Multiply = 16807;
        for (int i = 0; i < x.Length; i++)
            x[i] = random.NextDouble();
    }
}
```

---

**Tests of Goodness of Fit**

---

*KolmogorovOneSample • 795*
ICdfFunction cdf = new KolmogorovOneSampleEx1();
KolmogorovOneSample kos = new KolmogorovOneSample(cdf, x);
Console.WriteLine("D = " + kos.TestStatistic);
Console.WriteLine("D+ = " + kos.MaximumDifference);
Console.WriteLine("D- = " + kos.MinimumDifference);
Console.WriteLine("Z = " + kos.Z);
Console.WriteLine("Prob greater D one sided = " +
   kos.OneSidedPValue);
Console.WriteLine("Prob greater D two sided = " +
   kos.TwoSidedPValue);
Console.WriteLine("N missing = " + kos.NumberMissing);
}

public double CdfFunction(double x)
{
   double mean = 0.5;
   double std = 0.2886751;
   double z = (x - mean) / std;
   return Cdf.Normal(z);
}

Output
D = 0.121914080665685
D+ = 0.121914080665685
D- = 0.0869429640568776
Z = 1.21914080665685
Prob greater D one sided = 0.0511696542584409
Prob greater D two sided = 0.102339308516882
N missing = 0

KolmogorovTwoSample Class

public class Imsl.Stat.KolmogorovTwoSample

Performs a Kolmogorov-Smirnov two-sample test.

Class KolmogorovTwoSample computes Kolmogorov-Smirnov two-sample test statistics for testing that two continuous cumulative distribution functions (CDF’s) are identical based upon two random samples. One- or two-sided alternatives are allowed. Exact p-values are computed for the two-sided test when \(nm \leq 104\), where \(n\) is the number of non-missing \(X\) observations and \(m\) the number of non-missing \(Y\) observation.

Let \(F_n(x)\) denote the empirical CDF in the \(X\) sample, let \(G_m(y)\) denote the empirical CDF in the \(Y\) sample and let the corresponding population distribution functions be denoted by \(F(x)\) and \(G(y)\), respectively.
Then, the hypotheses tested by KolmogorovTwoSample are as follows:

\[ H_0 : F(x) = G(x) \quad H_1 : F(x) \neq G(x) \]
\[ H_0 : F(x) \geq G(x) \quad H_1 : F(x) < G(x) \]
\[ H_0 : F(x) \leq G(x) \quad H_1 : F(x) > G(x) \]

The test statistics are given as follows:

\[ D_{mn} = \max(D_{mn}^+, D_{mn}^-) \]
\[ D_{mn}^+ = \max_x(F_n(x) - G_m(x)) \]
\[ D_{mn}^- = \max_x(G_m(x) - F_n(x)) \]

Asymptotically, the distribution of the statistic

\[ Z = D_{mn} \sqrt{\frac{m+n}{mn}} \]

converges to a distribution given by Smirnov (1939).

Exact probabilities for the two-sided test are computed when \( mn \leq 104 \), according to an algorithm given by Kim and Jennrich (1973). When \( mn > 104 \), the very good approximations given by Kim and Jennrich are used to obtain the two-sided \( p \)-values. The one-sided probability is taken as one half the two-sided probability. This is a very good approximation when the \( p \)-value is small (say, less than 0.10) and not very good for large \( p \)-values.

### Properties

**MaximumDifference**

```csharp
public double MaximumDifference { get; }
```

**Description**

\( D^+ \), the maximum difference between the theoretical and empirical CDF’s.

**MinimumDifference**

```csharp
public double MinimumDifference { get; }
```

**Description**

\( D^- \), the minimum difference between the theoretical and empirical CDF’s.

**NumberMissingX**

```csharp
public int NumberMissingX { get; }
```

**Description**

Returns the number of missing values in the \( x \) sample.

**NumberMissingY**

```csharp
public int NumberMissingY { get; }
```

Tests of Goodness of Fit  
KolmogorovTwoSample  
797
Description
The number of missing values in the y sample.

OneSidedPValue
public double OneSidedPValue {get; }

Description
Probability of the statistic exceeding $D$ under the null hypothesis of equality and against the one-sided alternative. An exact probability is computed if the number of observation is less than or equal to 80, otherwise an approximate probability is computed.

TestStatistic
public double TestStatistic {get; }

Description
The test statistic, $D = \max(D^+, D^-)$.

TwoSidedPValue
public double TwoSidedPValue {get; }

Description
Probability of the statistic exceeding $D$ under the null hypothesis of equality and against the two-sided alternative. This probability is twice the probability, $p_1$, reported by OneSidedPValue, (or 1.0 if $p_1 \geq 1/2$). This approximation is nearly exact when $p_1 < 0.1$.

Z
public double Z {get; }

Description
The normalized $D$ statistic without the continuity correction applied.

Constructor

KolmogorovTwoSample
public KolmogorovTwoSample(double[] x, double[] y)

Description
Constructs a two sample Kolmogorov-Smirnov goodness-of-fit test.

Parameters
- x – Array containing the observations from the first sample.
- y – Array containing the observations from the second sample.
Example: Kolmogorov Two Sample

The following example illustrates the class `KolmogorovTwoSample` routine with two randomly generated samples from a uniform(0,1) distribution. Since the two theoretical distributions are identical, we would not expect to reject the null hypothesis.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;

public class KolmogorovTwoSampleEx1
{
    public static void Main(string[] args)
    {
        double[] x = new double[100];
        double[] y = new double[60];
        random.Multiplier = 16807;
        for (int i = 0; i < x.Length; i++)
        {
            x[i] = random.NextFloat();
        }
        for (int i = 0; i < y.Length; i++)
        {
            y[i] = random.NextFloat();
        }
        KolmogorovTwoSample k2s = new KolmogorovTwoSample(x, y);
        Console.WriteLine("D = " + k2s.TestStatistic);
        Console.WriteLine("D+ = " + k2s.MaximumDifference);
        Console.WriteLine("D- = " + k2s.MinimumDifference);
        Console.WriteLine("Z = " + k2s.Z);
        Console.WriteLine("Prob greater D one sided = " + k2s.OneSidedPValue);
        Console.WriteLine("Prob greater D two sided = " + k2s.TwoSidedPValue);
        Console.WriteLine("Missing X = " + k2s.NumberMissingX);
        Console.WriteLine("Missing Y = " + k2s.NumberMissingY);
    }
}
```

Output

D = 0.18  
D+ = 0.18  
D- = 0.01  
Z = 1.10227038425243  
Prob greater D one sided = 0.072010607348685  
Prob greater D two sided = 0.14402121469737  
Missing X = 0  
Missing Y = 0

Chapter 17. Tests of Goodness of Fit  
KolmogorovTwoSample • 799
Chapter 18: Time Series and Forecasting

Types

class AutoCorrelation ................................................................. 804
enumeration AutoCorrelation.StdErr ................................................. 813
class CrossCorrelation ................................................................. 813
enumeration CrossCorrelation.StdErr ............................................... 826
class MultiCrossCorrelation ............................................................. 826
class LackOfFit ............................................................................. 838
class ARAutoUnivariate ................................................................. 841
enumeration ARAutoUnivariate.ParamEstimation ............................... 862
class ARSeasonalFit ....................................................................... 862
enumeration ARSeasonalFit.CenterMethod ......................................... 873
class ARMA .................................................................................. 874
enumeration ARMA.ParamEstimation ............................................... 895
class ARMAEstimateMissing ............................................................... 896
enumeration ARMAEstimateMissing.MissingValueEstimation .............. 905
class ARMAPeakLikelihood ................................................................. 906
class ARMAOutlierIdentification ......................................................... 919
class AutoARIMA .......................................................................... 940
enumeration AutoARIMA.InformationCriterion .................................. 964
class Difference ............................................................................ 965
class GARCH ................................................................................ 970
class KalmanFilter ......................................................................... 976

Usage Notes

The classes in this chapter allow users to filter and analyze time series data using autoregressive, ARIMA, GARCH and state-space models. Some are designed for automatic model selection, estimation and forecasting using algorithms based upon minimizing AIC (Akaike’s Information Criteria).
General Methodology

Model Identification and Data Filtering

All classes in this chapter for time series analysis assume the data are stationary and collected at equally spaced times with no missing observations in the series. In order to prepare a series for analysis, any missing values must be replaced with estimates. The class ARMAEstimateMissing can be used to automatically estimate missing values using one of four, user-selected methods, provided the largest time gap with missing values has no more than 4 missing values.

In addition to estimating missing values, if a series is nonstationary or contains seasonal variation, its values should be filtered before fitting a model and preparing forecasts. Class ARSeasonalFit evaluates seasonal adjustments prior to modeling the series. Users specify a range of adjustments. ARSeasonalFit evaluates each adjustment to identify the one that minimizes the AIC. If a user already knows the series needs to be filtered using differences between consecutive values, the class Difference can be used to filter a series into an equivalent series of differences.

There are several classes available for transforming a time series into its autocorrelation matrix: AutoCorrelation, CrossCorrelation and MultiCrossCorrelation. Class AutoCorrelation computes the autocorrelation and partial autocorrelation matrices from a time series or its filtered values. Box and Jenkins (1976) describe how to identify the structure of an ARIMA model using the autocorrelation and partial autocorrelation matrices.

Classes CrossCorrelation and MultiCrossCorrelation are used to calculate the cross-correlation matrix for two univariate or multivariate time series, respectively.

Model Estimation and Forecasting

There are several classes useful for modeling stationary, equally spaced time series - ARMA, ARMAMaxLikelihood, GARCH, and KalmanFilter. Class ARMA can be used to fit autoregressive, moving-average and ARIMA (autoregressive, integrated moving average) models. Rather than passing the original series to this class, users are required to construct this class using the autocorrelation matrix of the series. Class AutoCorrelation is useful for computing this matrix prior to constructing the ARMA class.

The other classes require users to construct the class using the original series or its filtered values. Class ARMAMaxLikelihood estimates the maximum likelihood estimates of the ARMA parameters and prepares forecasts from these values. Class ARAutoUnivariate selects the best autoregressive model for an equally spaced, stationary time series using AIC. Class GARCH estimates the parameters of a GARCH model and the KalmanFilter class estimates the parameters for a state-space model using Kalman filtering and calculates one-step ahead forecasts.

Classes ARMA, ARMAMaxLikelihood and ARAutoUnivariate are the only classes with methods for obtaining forecasts from an ARIMA model. Class KalmanFilter can be used to obtain one-step ahead forecasts for a state-space model.

ARIMA Model (Autoregressive Integrated Moving Average)

A small, yet comprehensive, class of stationary time-series models consists of the nonseasonal ARMA processes defined by
\[ \phi(B)(W_t - \mu) = \theta(B)A_t, \quad t \in Z \]

where \( Z = \ldots, -2, -1, 0, 1, 2, \ldots \) denotes the set of integers, \( B \) is the backward shift operator defined by \( B^k W_t = W_{t-k} \), \( \mu \) is the mean of \( W_t \), and the following equations are true:

\[ \phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p, \quad p \geq 0 \]

\[ \theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q, \quad q \geq 0 \]

The model is of order \((p, q)\) and is referred to as an ARMA \((p, q)\) model.

An equivalent version of the ARMA \((p, q)\) model is given by

\[ \phi(B)W_t = \theta_0 + \theta(B)A_t, \quad t \in Z \]

where \( \theta_0 \) is an overall constant defined by the following:

\[ \theta_0 = \mu \left( 1 - \sum_{i=1}^{p} \phi_i \right) \]

See Box and Jenkins (1976, pp. 92-93) for a discussion of the meaning and usefulness of the overall constant.

If the “raw” data, \( \{Z_t\} \), are homogeneous and nonstationary, then differencing using the Difference class induces stationarity, and the model is called ARIMA (AutoRegressive Integrated Moving Average). Parameter estimation is performed on the stationary time series \( W_t = \Delta^d Z_t \), where \( \Delta^d = (1 - B)^d \) is the backward difference operator with period 1 and order \( d \), \( d > 0 \).

There are two classes for estimating the parameters in an ARMA model and preparing forecasts: ARMA and ARMA\text{MaxLikelihood}. Class ARMA estimates model parameters using either method of moments or least squares. The method of moments is selected by default or by setting property \texttt{Method} to \texttt{MethodOfMoments}. Method of moment estimates are good initial values for obtaining the least-squares estimates by setting property \texttt{Method} to \texttt{LeastSquares}. Other initial estimates provided by the user can be used. The least-squares procedure can be used to compute conditional or unconditional least-squares estimates of the parameters, depending on the choice of the backcasting length. The parameter estimates from either the method of moments or least-squares procedures can be used in the \texttt{Forecast} method. The functions for preliminary parameter estimation, least-squares parameter estimation, and forecasting follow the approach of Box and Jenkins (1976, Programs 2-4, pp. 498-509).

Class ARMA\text{MaxLikelihood} estimates model parameters using maximum likelihood. Users can tailor the maximum likelihood algorithm by setting tolerances and providing initial estimates for the parameters. By default, ARMA\text{MaxLikelihood} uses method of moment estimates to initialize the parameters. However, this can be overridden with the \texttt{SetAR} method in ARMA\text{MaxLikelihood}. 

\[ \text{Time Series and Forecasting} \quad \text{•} \quad 803 \]
AutoCorrelation Class

public class Imsl.Stat.AutoCorrelation

Computes the sample autocorrelation function of a stationary time series.

AutoCorrelation estimates the autocorrelation function of a stationary time series given a sample of \( n \) observations \( \{X_t\} \) for \( t = 1, 2, \ldots, n \).

Let \( \hat{\mu} = \text{xmean} \) be the estimate of the mean \( \mu \) of the time series \( \{X_t\} \) where

\[
\hat{\mu} = \begin{cases} 
\mu & \text{for } \mu \text{ known} \\
\frac{1}{n} \sum_{t=1}^{n} X_t & \text{for } \mu \text{ unknown}
\end{cases}
\]

The autocovariance function \( \sigma(k) \) is estimated by

\[
\hat{\sigma}(k) = \frac{1}{n} \sum_{t=1}^{n-k} (X_t - \hat{\mu}) (X_{t+k} - \hat{\mu}), \quad k = 0, 1, \ldots, K
\]

where \( K = \text{maximumLag} \). Note that \( \hat{\sigma}(0) \) is an estimate of the sample variance. The autocorrelation function \( \rho(k) \) is estimated by

\[
\hat{\rho}(k) = \frac{\hat{\sigma}(k)}{\hat{\sigma}(0)}, \quad k = 0, 1, \ldots, K
\]

Note that \( \hat{\rho}(0) \equiv 1 \) by definition.

The standard errors of sample autocorrelations may be optionally computed according to the GetStandardErrors method argument stdErrMethod. One method (Bartlett 1946) is based on a general asymptotic expression for the variance of the sample autocorrelation coefficient of a stationary time series with independent, identically distributed normal errors. The theoretical formula is

\[
\text{var}(\hat{\rho}(k)) = \frac{1}{n} \sum_{i=-\infty}^{\infty} \left[ \rho^2(i) + \rho(i-k)\rho(i+k) - 4\rho(i)\rho(k) + 2\rho^2(i)\rho^2(k) \right]
\]

where \( \hat{\rho}(k) \) assumes \( \mu \) is unknown. For computational purposes, the autocorrelations \( \rho(k) \) are replaced by their estimates \( \hat{\rho}(k) \) for \( |k| \leq K \), and the limits of summation are bounded because of the assumption that \( \rho(k) = 0 \) for all \( k \) such that \( |k| > K \).

A second method (Moran 1947) utilizes an exact formula for the variance of the sample autocorrelation coefficient of a random process with independent, identically distributed normal errors. The theoretical formula is
\[ \text{var}\{\hat{\rho}(k)\} = \frac{n-k}{n(n+2)} \]

where \( \mu \) is assumed to be equal to zero. Note that this formula does not depend on the autocorrelation function.

The method `GetPartialAutoCorrelations` returns the estimated partial autocorrelations of the stationary time series given \( K = \text{maximumLag} \) sample autocorrelations \( \hat{\rho}(k) \) for \( k=0,1,\ldots,K \). Consider the AR(\( k \)) process defined by

\[
X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_k X_{t-k} + A_t
\]

where \( \phi_{kj} \) denotes the \( j \)-th coefficient in the process. The set of estimates \( \{\hat{\phi}_{kk}\} \) for \( k = 1, \ldots, K \) is the sample partial autocorrelation function. The autoregressive parameters \( \{\hat{\phi}_{kj}\} \) for \( j = 1, \ldots, k \) are approximated by Yule-Walker estimates for successive AR(\( k \)) models where \( k = 1, \ldots, K \). Based on the sample Yule-Walker equations

\[
\hat{\rho}(j) = \hat{\phi}_{k1}\hat{\rho}(j-1) + \hat{\phi}_{k2}\hat{\rho}(j-2) + \cdots + \hat{\phi}_{kk}\hat{\rho}(j-k), \quad j = 1, 2, \ldots, k
\]

a recursive relationship for \( k=1, \ldots, K \) was developed by Durbin (1960). The equations are given by

\[
\hat{\phi}_{kk} = \begin{cases} 
\hat{\rho}(1) & \text{for } k = 1 \\
\hat{\rho}(k) - \sum_{j=1}^{k-1} \hat{\phi}_{kj} \hat{\rho}(j-k) & \text{for } k = 2, \ldots, K
\end{cases}
\]

and

\[
\hat{\phi}_{kj} = \begin{cases} 
\hat{\phi}_{k-1,j} - \hat{\phi}_{kk}\hat{\phi}_{k-1,k-j} & \text{for } j = 1, 2, \ldots, k-1 \\
\hat{\phi}_{kk} & \text{for } j = k
\end{cases}
\]

This procedure is sensitive to rounding error and should not be used if the parameters are near the nonstationarity boundary. A possible alternative would be to estimate \( \{\hat{\phi}_{kk}\} \) for successive AR(\( k \)) models using least or maximum likelihood. Based on the hypothesis that the true process is AR(\( p \)), Box and Jenkins (1976, page 65) note

\[
\text{var}\{\hat{\phi}_{kk}\} \simeq \frac{1}{n} \quad k \geq p + 1
\]

See Box and Jenkins (1976, pages 82-84) for more information concerning the partial autocorrelation function.

**Properties**

- **Mean**

public double Mean {get; set; }

---

**Time Series and Forecasting**

AutoCorrelation • 805
Description
The mean of the time series $x$.

Property Value
A double containing the mean of the time series $x$.

**NumberOfProcessors**
public int NumberOfProcessors {get; set; }

Description
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

Property Value
An int indicating the maximum possible number of processors to use.

Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**Variance**
public double Variance {get; }

Description
Returns the variance of the time series $x$.

Property Value
A double containing the variance of the time series $x$.

**Constructor**

**AutoCorrelation**
public AutoCorrelation(double[] $x$, int maximumLag)

Description
Constructor to compute the sample autocorrelation function of a stationary time series.

Parameters
$x$ – A one-dimensional double array containing the stationary time series.
maximumLag – An int containing the maximum lag of autocovariance, autocorrelations, and standard errors of autocorrelations to be computed.

Remarks
maximumLag must be greater than or equal to 1 and less than the number of observations in $x$. 
Methods

GetAutoCorrelations
public double[] GetAutoCorrelations()

Description
Returns the autocorrelations of the time series x.

Returns
A double array of length maximumLag +1 containing the autocorrelations of the time series x.

Remarks
The 0-th element of this array is 1. The k-th element of this array contains the autocorrelation of lag k where k = 1, ..., maximumLag.

GetAutoCovariances
public double[] GetAutoCovariances()

Description
Returns the variance and autocovariances of the time series x.

Returns
A double array of length maximumLag + 1 containing the variances and autocovariances of the time series x.

Remarks
The 0-th element of the array contains the variance of the time series x. The k-th element contains the autocovariance of lag k where k = 1, ..., maximumLag.

Exception
Imsl.Stat.NonPosVarianceException is thrown if the problem is ill-conditioned.

GetPartialAutoCorrelations
public double[] GetPartialAutoCorrelations()

Description
Returns the sample partial autocorrelation function of the stationary time series x.

Returns
A double array of length maximumLag containing the partial autocorrelations of the time series x.

GetStandardErrors

Description
Returns the standard errors of the autocorrelations of the time series x.
Parameter

stderrMethod – An int specifying the method to compute the standard errors of autocorrelations of the time series \( x \).

Returns

A double array of length maximumLag containing the standard errors of the autocorrelations of the time series \( x \).

Remarks

Method of computation for standard errors of the autocorrelation is chosen by the stderrMethod parameter.

If stderrMethod is set to Bartletts, Bartlett’s formula is used to compute the standard errors of autocorrelations.

If stderrMethod is set to Morana, Moran’s formula is used to compute the standard errors of autocorrelations.

Example 1: AutoCorrelation

Consider the Wolfer Sunspot Data (Anderson 1971, p. 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869. This example computes the estimated autocovariances, estimated autocorrelations, and estimated standard errors of the autocorrelations using both Barlett and Moran formulas.

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class AutoCorrelationEx1
{
    public static void Main(String[] args)
    {
        double[] x = new double[]
        { 100.8, 81.6, 66.5, 34.8, 30.6,
          7, 19.8, 92.5, 154.4, 125.9,
          84.8, 68.1, 38.5, 22.8, 10.2,
          24.1, 82.9, 132, 130.9, 118.1,
          89.9, 66.6, 60, 46.9, 41,
          21.3, 16, 6.4, 4.1, 6.8,
          14.5, 34, 45, 43.1, 47.5,
          42.2, 28.1, 10.1, 8.1, 2.5,
          0, 1.4, 5, 12.2, 13.9,
          35.4, 45.8, 41.1, 30.4, 23.9,
          15.7, 6.6, 4, 1.8, 8.5,
          16.6, 36.3, 49.7, 62.5, 67,
          71, 47.8, 27.5, 8.5, 13.2,
          56.9, 121.5, 138.3, 103.2, 85.8,
          63.2, 36.8, 24.2, 10.7, 15,
          40.1, 61.5, 98.5, 124.3, 95.9,
          66.5, 64.5, 54.2, 39, 20.6,
          6.7, 4.3, 22.8, 54.8, 93.8,
```
AutoCorrelation ac = new AutoCorrelation(x, 20);

new PrintMatrix
    ("AutoCovariances are: ").Print(ac.GetAutoCovariances());
Console.Out.WriteLine();
new PrintMatrix
    ("AutoCorrelations are: ").Print(ac.GetAutoCorrelations());
Console.Out.WriteLine();
Console.Out.WriteLine("Mean = " + ac.Mean);
new PrintMatrix
    ("Standard Error using Bartlett are: ").Print
    (ac.GetStandardErrors(AutoCorrelation.StdErr.Bartletts));
Console.Out.WriteLine();
new PrintMatrix
    ("Standard Error using Moran are: ").Print
    (ac.GetStandardErrors(AutoCorrelation.StdErr.Morans));
Console.Out.WriteLine();
new PrintMatrix
    ("Partial AutoCovariances: ").Print
    (ac.GetPartialAutoCorrelations());
ac.Mean = 50;
new PrintMatrix
    ("AutoCovariances are: ").Print
    (ac.GetAutoCovariances());
Console.Out.WriteLine();
new PrintMatrix
    ("AutoCorrelations are: ").Print
    (ac.GetAutoCorrelations());
Console.Out.WriteLine();
new PrintMatrix
    ("Standard Error using Bartlett are: ").Print
    (ac.GetStandardErrors(AutoCorrelation.StdErr.Bartletts));
}

Output
AutoCovariances are:
   0 1382.908024
   1 1115.02915024
   2 592.00446848
   3 95.29741072
   4 -235.95179904
   5 -370.0108088
   6 -294.25541456
   7 -60.44237232
   8 227.63259792
   9 458.38076816
  10 567.8407384
  11 546.12202864
  12 398.93728688
AutoCorrelations are:

0  1
1  0.806293065691258
2  0.428086653780237
3  0.0689108813212006
4  -0.170620023128885
5  -0.26759955093586
6  -0.212780177317129
7  -0.043706718936501
8  0.164604293249802
9  0.331461500117813
10  0.41061524938228
11  0.394908424249623
12  0.288477093166393
13  0.143001143740562
14  0.0194453129877855
15  -0.055827637549379
16  -0.10393518127421
17  -0.146103713633525
18  -0.17743206681559
19  -0.166906019369514
20  -0.103917661565611

Mean = 46.976

Standard Error using Bartlett are:

0  1
1  0.0347838253702384
2  0.0962419914340011
3  0.156783378574532
4  0.205766777086907
5  0.2309565775779118
6  0.228994712235613
7  0.208621905639667
8  0.178475936561125
9  0.145727084432033
10  0.134405581638002
11  0.150675803916788
12  0.174348147103935
13  0.190619474229408
14  0.195490061669564
15  0.195892530944597
16  0.196285328179458
17  0.196020624500033
18  0.198716030900604
19  0.198716030900604
20  0.198716030900604
Standard Error using Moran are:

0
1 0.0985184366143778
2 0.0980196058819607
3 0.0975182235357506
4 0.0970142500145332
5 0.0965076447241154
6 0.0959983659999659
7 0.0954863710632231
8 0.0949716159886734
9 0.094454055643212
10 0.09393364366272
11 0.0934103323839415
12 0.0928840728025648
13 0.0923548145182799
14 0.0918225056781811
15 0.0912870929175277
16 0.090748521297303
17 0.0902067342384192
18 0.0896616734523426
19 0.0891132788679007

Partial AutoCovariances:

0
1 -0.634544877310468
2 0.0792508772709519
3 -0.0585660846582815
4 -0.0099021571933657
5 0.171719898229681
6 0.108591873581717
7 0.11000138764865
8 0.0785374339029981
9 0.079156332964613
10 0.0687065876031485
11 -0.0378019674610775
12 0.0811184838397538
13 0.0334122414991749
14 -0.034867839607946
15 -0.130648157884444
16 -0.154900984829049
17 -0.119085063160732
18 -0.0161889037437313
19 -0.0038517549253345

AutoCovariances are:

0
1 1392.0526
2 1126.5241
3 604.1624
AutoCorrelations are:

0
0 1
1 0.809253975029392
2 0.434008312616923
3 0.0766885532917362
4 -0.162265420142888
5 -0.259347886710603
6 -0.205861545749061
7 -0.0386194458456527
8 0.16950975846746
9 0.338195338308337
10 0.49534649768263
11 0.405705861976767
12 0.300536847530043
13 0.155241547625427
14 0.0309794328174093
15 -0.04559332025241
16 -0.0944656832651295
17 -0.135815773053403
18 -0.165000086922003
19 -0.15240508715757
20 -0.0873309672350025

Standard Error using Bartlett are:

0
0 0.0344591054641365
1 0.0972222809088609
2 0.15947410033087
3 0.209799660647689
4 0.235599778243579
5 0.233236443705991
6 0.211657508693781
7 0.180412936841618
8 0.14689653606348

812  •  AutoCorrelation

IMSL C# Numerical Library
AutoCorrelation.StdErr Enumeration

public enumeration Imsl.Stat.AutoCorrelation.StdErr

Standard Error computation method.

Fields

Bartletts
public Imsl.Stat.AutoCorrelation.StdErr Bartletts

Description
Indicates standard error computation using Bartlett’s formula.

Morans
public Imsl.Stat.AutoCorrelation.StdErr Morans

Description
Indicates standard error computation using Moran’s formula.

CrossCorrelation Class

public class Imsl.Stat.CrossCorrelation

Computes the sample cross-correlation function of two stationary time series.
CrossCorrelation estimates the cross-correlation function of two jointly stationary time series given a sample of \( n = x.\) Length observations \( \{X_t\} \) and \( \{Y_t\} \) for \( t = 1,2,\ldots,n.\)

Let
\[
\hat{\mu}_x = \text{xmean}
\]
be the estimate of the mean \( \mu_X \) of the time series \( \{X_t\} \) where
\[
\hat{\mu}_X = \begin{cases} 
\mu_X & \text{for } \mu_X \text{ known} \\
\frac{1}{n} \sum_{t=1}^{n} X_t & \text{for } \mu_X \text{ unknown}
\end{cases}
\]

The autocovariance function of \( \{X_t\}, \sigma_X(k), \) is estimated by
\[
\hat{\sigma}_X(k) = \frac{1}{n} \sum_{t=1}^{n-k} (X_t - \hat{\mu}_X)(X_{t+k} - \hat{\mu}_X), \quad k = 0,1,\ldots,K
\]
where \( K = \text{maximumLag}. \) Note that \( \hat{\sigma}_X(0) \) is equivalent to the sample variance of \( x \) returned by property \( \text{VarianceX}. \) The autocorrelation function \( \rho_X(k) \) is estimated by
\[
\hat{\rho}_X(k) = \frac{\hat{\sigma}_X(k)}{\hat{\sigma}_X(0)}, \quad k = 0,1,\ldots,K
\]

Note that \( \hat{\rho}_x(0) = 1 \) by definition. Let
\[
\hat{\mu}_Y = \text{ymean}, \hat{\sigma}_Y(k), \text{and} \hat{\rho}_Y(k)
\]
be similarly defined.

The cross-covariance function \( \sigma_{XY}(k) \) is estimated by
\[
\hat{\sigma}_{XY}(k) = \begin{cases} 
\frac{1}{n} \sum_{t=1}^{n-k} (X_t - \hat{\mu}_X)(Y_{t+k} - \hat{\mu}_Y) & k = 0,1,\ldots,K \\
\frac{1}{n} \sum_{t=1}^{n} (X_t - \hat{\mu}_X)(Y_{t+k} - \hat{\mu}_Y) & k = -1,-2,\ldots,-K
\end{cases}
\]

The cross-correlation function \( \rho_{XY}(k) \) is estimated by
\[
\hat{\rho}_{XY}(k) = \frac{\hat{\sigma}_{XY}(k)}{[\hat{\sigma}_X(0)\hat{\sigma}_Y(0)]^{1/2}}, \quad k = 0,\pm 1,\ldots,\pm K
\]

The standard errors of the sample cross-correlations may be optionally computed according to the \text{GetStandardErrors} method argument \text{stderrMethod}. One method is based on a general asymptotic expression for the variance of the sample cross-correlation coefficient of two jointly stationary time series with independent, identically distributed normal errors given by Bartlet (1978, page 352). The theoretical formula is
\[
\text{var} \{\hat{\rho}_{XY}(k)\} = \frac{1}{n-1} \sum_{i=-\infty}^{\infty} \left[ \rho_X(i) + \rho_{XY}(i-k)\rho_{XY}(i+k) \right. \\
-2\rho_{XY}(k)\{\rho_X(i)\rho_{XY}(i+k) + \rho_X(-i)\rho_Y(i+k)\} \\
+ \rho_X^2(i)\{\rho_X(i) + \frac{1}{2}\rho_Y^2(i) + \frac{1}{4}\rho_Y^2(i)\}
\]

814 • CrossCorrelation

IMSL C# Numerical Library
For computational purposes, the autocorrelations $\rho_X(k)$ and $\rho_Y(k)$ and the cross-correlations $\rho_{XY}(k)$ are replaced by their corresponding estimates for $|k| \leq K$, and the limits of summation are equal to zero for all $k$ such that $|k| > K$.

A second method evaluates Bartlett’s formula under the additional assumption that the two series have no cross-correlation. The theoretical formula is

$$\text{var}\{\hat{\rho}_{XY}(k)\} = \frac{1}{n-k} \sum_{i=-\infty}^{\infty} \rho_X(i)\rho_Y(i) \quad k \geq 0$$

For additional special cases of Bartlett’s formula, see Box and Jenkins (1976, page 377).

An important property of the cross-covariance coefficient is $\sigma_{XY}(k) = \sigma_{YX}(-k)$ for $k \geq 0$. This result is used in the computation of the standard error of the sample cross-correlation for lag $k < 0$. In general, the cross-covariance function is not symmetric about zero so both positive and negative lags are of interest.

**Properties**

**MeanX**
public double MeanX {get; set; }

Description
Estimate of the mean of time series $x$.

**Property Value**
A double containing the estimate of the mean of time series $x$.

**MeanY**
public double MeanY {get; set; }

Description
Estimate of the mean of time series $y$.

**Property Value**
A double containing the estimate mean of the time series $y$.

**NumberOfProcessors**
public int NumberOfProcessors {get; set; }

Description
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**
An int indicating the maximum possible number of processors to use.
Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

VarianceX
public double VarianceX {get; }

Description
Returns the variance of time series x.

Property Value
A double containing the variance of the time series x.

Exception
Imsl.Stat.NonPosVarianceXYException is thrown if the problem is ill-conditioned. The variance is too small to work with.

VarianceY
public double VarianceY {get; }

Description
Returns the variance of time series y.

Property Value
A double containing the variance of the time series y.

Exception
Imsl.Stat.NonPosVarianceXYException is thrown if the problem is ill-conditioned. The variance is too small to work with.

Constructor

CrossCorrelation
public CrossCorrelation(double[] x, double[] y, int maximumLag)

Description
Constructor to compute the sample cross-correlation function of two stationary time series.

Parameters
x – A one-dimensional double array containing the first stationary time series.
y – A one-dimensional double array containing the second stationary time series.
maximumLag – An int containing the maximum lag of the cross-covariance and cross-correlations to be computed.

816 • CrossCorrelation  IMSL C# Numerical Library
Remarks
maximumLag must be greater than or equal to 1 and less than the minimum of the number of observations of x and y.

Methods

GetAutoCorrelationX
public double[] GetAutoCorrelationX()

Description
Returns the autocorrelations of the time series x.

Returns
A double array of length maximumLag + 1 containing the autocorrelations of the time series x.

Remarks
The 0-th element of this array is 1. The k-th element of this array contains the autocorrelation of lag k where k = 1, ..., maximumLag.

Exception
Imsl.Stat.NonPosVarianceException is thrown if the problem is ill-conditioned.

GetAutoCorrelationY
public double[] GetAutoCorrelationY()

Description
Returns the autocorrelations of the time series y.

Returns
A double array of length maximumLag + 1 containing the autocorrelations of the time series y.

Remarks
The 0-th element of this array is 1. The k-th element of this array contains the autocorrelation of lag k where k = 1, ..., maximumLag.

Exception
Imsl.Stat.NonPosVarianceException is thrown if the problem is ill-conditioned.

GetAutoCovarianceX
public double[] GetAutoCovarianceX()

Description
Returns the autocovariances of the time series x.

Returns
A double array of length maximumLag + 1 containing the variances and autocovariances of the time series x.
Remarks
The 0-th element of the array contains the variance of the time series x. The k-th elements contains the autocovariance of lag k where k = 1, ..., maximumLag.

Exception
Imsl.Stat.NonPosVarianceException is thrown if the problem is ill-conditioned.

GetAutoCovarianceY
public double[] GetAutoCovarianceY()

Description
Returns the autocovariances of the time series y.

Returns
A double array of length maximumLag + 1 containing the variances and autocovariances of the time series y.

Remarks
The 0-th element of the array contains the variance of the time series y. The k-th elements contains the autocovariance of lag k where k = 1, ..., maximumLag.

Exception
Imsl.Stat.NonPosVarianceException is thrown if the problem is ill-conditioned.

GetCrossCorrelations
public double[] GetCrossCorrelations()

Description
Returns the cross-correlations between the time series x and y.

Returns
A double array of length 2 * maximumLag + 1 containing the cross-correlations between the time series x and y.

Remarks
The cross-correlation between x and y at lag k, where k = -maximumLag, ..., 0, 1, ..., maximumLag, corresponds to output array indices 0, 1, ..., (2*maximumLag).

Exception
Imsl.Stat.NonPosVarianceXYException is thrown if the problem is ill-conditioned. The variance is too small to work with.

GetCrossCovariances
public double[] GetCrossCovariances()

Description
Returns the cross-covariances between the time series x and y.
Returns
A double array of length \(2 \times \text{maximumLag} + 1\) containing the cross-covariances between the time series \(x\) and \(y\).

Remarks
The cross-covariance between \(x\) and \(y\) at lag \(k\), where \(k = -\text{maximumLag},..., 0, 1,...,\text{maximumLag}\), corresponds to output array indices \(0, 1,..., (2\times\text{maximumLag})\).

GetStandardErrors

Description
Returns the standard errors of the cross-correlations between the time series \(x\) and \(y\).

Parameter
- stderrMethod – An \(\text{int}\) specifying the method to compute the standard errors of cross-correlations between the time series \(x\) and \(y\).

Returns
A double array of length \(2 \times \text{maximumLag} + 1\) containing the standard errors of the cross-correlations between the time series \(x\) and \(y\).

Remarks
The standard error of cross-correlations between \(x\) and \(y\) at lag \(k\), where \(k = -\text{maximumLag},..., 0, 1,...,\text{maximumLag}\), corresponds to output array indices \(0, 1,..., (2\times\text{maximumLag})\).

Method of computation for standard errors of the cross-correlation is determined by the stderrMethod parameter. If stderrMethod is set to Bartletts, Bartlett’s formula is used to compute the standard errors of cross-correlations. If stderrMethod is set to BartlettsNoCC, Bartlett’s formula is used to compute the standard errors of cross-correlations, with the assumption of no cross-correlation.

Exception
- Imsl.Stat.NonPosVarianceException is thrown if the problem is ill-conditioned.

Example 1: CrossCorrelation
Consider the Gas Furnace Data (Box and Jenkins 1976, pages 532-533) where \(X\) is the input gas rate in cubic feet/minute and \(Y\) is the percent \(CO_2\) in the outlet gas. The CrossCorrelation methods GetCrossCovariance and GetCrossCorrelation are used to compute the cross-covariances and cross-correlations between time series \(X\) and \(Y\) with lags from \(-\text{maximumLag} = -10\) through lag \(\text{maximumLag} = 10\). In addition, the estimated standard errors of the estimated cross-correlations are computed. In the first invocation of method GetStandardErrors stderrMethod = Bartletts, the standard errors are based on the assumption that autocorrelations and cross-correlations for lags greater than \(\text{maximumLag}\) or less than \(-\text{maximumLag}\) are zero. In the second invocation of method GetStandardErrors with stderrMethod = BartlettsNoCC, the standard errors are based on the additional assumption that all cross-correlations for \(X\) and \(Y\) are zero.
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class CrossCorrelationEx1
{
    public static void Main(String[] args)
    {
        double[] x2 = new double[] {100.8, 81.6, 66.5, 34.8, 30.6,
                                    7, 19.8, 92.5, 154.4, 125.9,
                                    84.8, 68.1, 38.5, 22.8, 10.2,
                                    24.1, 82.9, 132, 130.9, 118.1,
                                    89.9, 66.6, 60, 46.9, 41,
                                    21.3, 16.4, 4.1, 6.8,
                                    14.5, 34, 45, 43.1, 47.5,
                                    42.2, 28.1, 10.1, 8.1, 2.5,
                                    0, 1.4, 5, 12.2, 13.9,
                                    35.4, 45.8, 41.1, 30.4, 23.9,
                                    15.7, 6.6, 4, 1.8, 8.5,
                                    16.6, 36.3, 49.7, 62.5, 67,
                                    71, 47.8, 27.5, 8.5, 13.2,
                                    56.9, 121.5, 138.3, 103.2, 85.8,
                                    63.2, 36.8, 24.2, 10.7, 15,
                                    40.1, 61.5, 98.5, 124.3, 95.9,
                                    66.5, 64.5, 54.2, 39, 20.6,
                                    6.7, 4.3, 22.8, 54.8, 93.8,
                                    95.7, 77.2, 59.1, 44, 47,
                                    30.5, 16.3, 7.3, 37.3, 73.9};

        double[] x = new double[] {-0.109, 0.0, 0.178, 0.339, 0.373,
                                   0.441, 0.461, 0.348, 0.127,
                                   -0.18, -0.588, -1.055, -1.421,
                                   -1.52, -1.302, -0.814, -0.475,
                                   -0.193, 0.088, 0.435, 0.771,
                                   0.866, 0.875, 0.891, 0.987, 1.263,
                                   1.775, 1.976, 1.934, 1.866, 1.832,
                                   1.767, 1.608, 1.265, 0.79, 0.36,
                                   0.115, 0.088, 0.331, 0.645, 0.96,
                                   1.409, 2.67, 2.834, 2.812, 2.483,
                                   1.929, 1.485, 1.214, 1.239, 1.608,
                                   1.905, 2.023, 1.815, 0.535, 0.122,
                                   0.009, 0.164, 0.671, 1.019, 1.146,
                                   1.155, 1.112, 1.121, 1.223, 1.257,
                                   1.157, 0.913, 0.62, 0.255, -0.28,
                                   -1.08, -1.551, -1.799, -1.825,
                                   -1.456, -0.944, -0.57, -0.431,
                                   -0.577, -0.96, -1.616, -1.875,
                                   -1.891, -1.746, -1.474, -1.201,
                                   -0.927, -0.524, 0.04, 0.788,
                                   0.943, 0.93, 1.006, 1.137, 1.198,
                                   1.054, 0.595, -0.08, -0.314,
                                   -0.288, -0.153, -0.109, -0.187,
                                   -0.255, -0.229, -0.007, 0.254,
                                   0.33, 0.102, -0.423, -1.139,
                                   -2.275, -2.594, -2.716, -2.51,
                                   -1.79, -1.346, -1.081, -0.91,
                                   -0.876, -0.885, -0.8, -0.544,

        CrossCorrelation IMSL C# Numerical Library
- 0.416, - 0.271, 0.0, 0.403, 0.841, 1.285, 1.607, 1.746, 1.683, 1.485, 0.993, 0.648, 0.577, 0.577, 0.632, 0.747, 0.9, 0.993, 0.968, 0.79, 0.399, - 0.161, - 0.553, - 0.603, - 0.424, - 0.194, - 0.049, 0.06, 0.161, 0.301, 0.517, 0.566, 0.56, 0.573, 0.592, 0.671, 0.933, 1.337, 1.46, 1.353, 0.772, 0.218, - 0.237, - 0.714, - 1.099, - 1.269, - 1.175, - 0.676, 0.033, 0.556, 0.643, 0.484, 0.109, - 0.31, - 0.697, - 1.047, - 1.218, - 1.183, - 0.873, - 0.336, 0.063, 0.084, 0.0, 0.001, 0.209, 0.556, 0.782, 0.858, 0.918, 0.862, 0.416, - 0.336, - 0.959, - 1.813, - 2.378, - 2.499, - 2.473, - 2.33, - 2.053, - 1.739, - 1.261, - 0.569, - 0.137, - 0.024, - 0.05, - 0.135, - 0.276, - 0.534, - 0.871, - 1.243, - 1.439, - 1.422, - 1.175, - 0.813, - 0.634, - 0.582, - 0.625, - 0.713, - 0.848, - 1.039, - 1.346, - 1.628, - 1.619, - 1.149, - 0.488, - 0.16, - 0.007, - 0.092, - 0.62, - 1.086, - 1.525, - 1.858, - 2.029, - 2.024, - 1.961, - 1.952, - 1.794, - 1.302, - 1.03, - 0.918, - 0.798, - 0.867, - 1.047, - 1.123, - 0.876, - 0.395, 0.185, 0.662, 0.709, 0.605, 0.501, 0.603, 0.943, 1.223, 1.249, 0.824, 0.102, 0.025, 0.382, 0.922, 1.032, 0.866, 0.527, 0.093, - 0.458, - 0.748, - 0.947, - 1.029, - 0.928, - 0.645, - 0.424, - 0.276, - 0.158, - 0.033, 0.102, 0.251, 0.28, 0.0, - 0.493, - 0.759, - 0.824, - 0.74, - 0.528, - 0.204, 0.034, 0.204, 0.253, 0.195, 0.131, 0.017, - 0.182, - 0.262};

double[] y = new double[]{53.8, 53.6, 53.5, 53.5, 53.4, 53.1, 52.7, 52.4, 52.2, 52.0, 52.0, 52.4, 53.0, 54.0, 54.9, 56.0, 56.8, 56.8, 56.4, 55.7, 55.0, 54.3, 53.2, 52.3, 51.6, 51.2, 50.8, 50.5, 50.0, 49.2, 48.4, 47.9, 47.6, 47.5, 47.5, 47.6, 48.1, 49.0, 50.0, 51.1, 51.8, 51.9, 51.7, 51.2, 50.0, 48.3, 47.0, 45.8, 45.6, 46.0, 46.9, 47.8, 48.2, 48.3, 47.9, 47.2, 47.2, 48.1, 49.4, 50.6, 51.5, 51.6, 51.2, 50.5, 50.1, 49.8, 49.6, 49.4, 49.3, 49.2, 49.3, 49.7, 50.3, 51.3, 52.8, 54.4, 56.0, 56.9, 57.5, 57.3, 56.6,

Time Series and Forecasting

CrossCorrelation • 821
```
CrossCorrelation cc = new CrossCorrelation(x, y, 10);
Console.Out.WriteLine("Mean = " + cc.MeanX);
Console.Out.WriteLine("Mean = " + cc.MeanY);
Console.Out.WriteLine("Xvariance = " + cc.VarianceX);
Console.Out.WriteLine("Yvariance = " + cc.VarianceY);
new PrintMatrix
   ("CrossCovariances are: ").Print(cc.GetCrossCovariances());
new PrintMatrix
   ("CrossCorrelations are: ").Print(cc.GetCrossCorrelations());

double[] stdErrors =
   cc.GetStandardErrors(CrossCorrelation.StdErr.Bartletts);
```
new PrintMatrix
    ("Standard Errors using Bartlett are: ").Print(stdErrors);

stdErrors =
    cc.GetStandardErrors(CrossCorrelation.StdErr.BartlettsNoCC);
new PrintMatrix("Standard Errors using Bartlett #2 are: ").Print
    (stdErrors);

new PrintMatrix("AutoCovariances of X are: ").Print
    (cc.GetAutoCovarianceX());
new PrintMatrix("AutoCovariances of Y are: ").Print
    (cc.GetAutoCovarianceY());
new PrintMatrix("AutoCorrelations of X are: ").Print
    (cc.GetAutoCorrelationX());
new PrintMatrix("AutoCorrelations of Y are: ").Print
    (cc.GetAutoCorrelationY());
}
}

Output
Mean = -0.0568344594594595
Mean = 53.5091216216216
Xvariance = 1.14693790165038
Yvariance = 10.2189370662893
CrossCovariances are:
  0
  0 -0.404501563294314
  1 -0.508490782763824
  2 -0.614369467627782
  3 -0.705476130258359
  4 -0.776166564117932
  5 -0.831473609098764
  6 -0.891315326970392
  7 -0.980605209560792
  8 -1.12477059434257
  9 -1.34704305203341
 10 -1.65852650999917
 11 -2.04865124574232
 12 -2.4821658776478
 13 -2.88541054192018
 14 -3.16536049680239
 15 -3.25343758942199
 16 -3.3112860301494
 17 -2.83913985444463
 18 -2.4530218901565
 19 -2.05268794198849
 20 -1.694654617713
CrossCorrelations are:
  0
  0 -0.118153717307789
  1 -0.148528662561878
  2 -0.179455515102209
  3 -0.206067503381416

Time Series and Forecasting
CrossCorrelation • 823
<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-0.226715971265165</td>
</tr>
<tr>
<td>5</td>
<td>-0.242870996488244</td>
</tr>
<tr>
<td>6</td>
<td>-0.260350586329711</td>
</tr>
<tr>
<td>7</td>
<td>-0.286431898500946</td>
</tr>
<tr>
<td>8</td>
<td>-0.3285421835153</td>
</tr>
<tr>
<td>9</td>
<td>-0.39346731487308</td>
</tr>
<tr>
<td>10</td>
<td>-0.484450717109386</td>
</tr>
<tr>
<td>11</td>
<td>-0.59840506361053</td>
</tr>
<tr>
<td>12</td>
<td>-0.72503348897091</td>
</tr>
<tr>
<td>13</td>
<td>-0.84281935503927</td>
</tr>
<tr>
<td>14</td>
<td>-0.92459249205792</td>
</tr>
<tr>
<td>15</td>
<td>-0.95031953992448</td>
</tr>
<tr>
<td>16</td>
<td>-0.914593458680361</td>
</tr>
<tr>
<td>17</td>
<td>-0.829320215245049</td>
</tr>
<tr>
<td>18</td>
<td>-0.716520475473708</td>
</tr>
<tr>
<td>19</td>
<td>-0.599584112456951</td>
</tr>
<tr>
<td>20</td>
<td>-0.495003641096017</td>
</tr>
</tbody>
</table>

Standard Errors using Bartlett are:

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.158147783754555</td>
</tr>
<tr>
<td>1</td>
<td>0.155750271182418</td>
</tr>
<tr>
<td>2</td>
<td>0.152735096430409</td>
</tr>
<tr>
<td>3</td>
<td>0.149086745416716</td>
</tr>
<tr>
<td>4</td>
<td>0.145054990300008</td>
</tr>
<tr>
<td>5</td>
<td>0.141300099196058</td>
</tr>
<tr>
<td>6</td>
<td>0.138420534019813</td>
</tr>
<tr>
<td>7</td>
<td>0.136074039397204</td>
</tr>
<tr>
<td>8</td>
<td>0.132158917844376</td>
</tr>
<tr>
<td>9</td>
<td>0.123531347020305</td>
</tr>
<tr>
<td>10</td>
<td>0.107879045104545</td>
</tr>
<tr>
<td>11</td>
<td>0.0873410658167485</td>
</tr>
<tr>
<td>12</td>
<td>0.064140795847026</td>
</tr>
<tr>
<td>13</td>
<td>0.0469456102701398</td>
</tr>
<tr>
<td>14</td>
<td>0.0440970262220149</td>
</tr>
<tr>
<td>15</td>
<td>0.0482335854893665</td>
</tr>
<tr>
<td>16</td>
<td>0.0491545707033738</td>
</tr>
<tr>
<td>17</td>
<td>0.0475621871011123</td>
</tr>
<tr>
<td>18</td>
<td>0.0534780426650682</td>
</tr>
<tr>
<td>19</td>
<td>0.0715660938138719</td>
</tr>
<tr>
<td>20</td>
<td>0.0939330263600716</td>
</tr>
</tbody>
</table>

Standard Errors using Bartlett #2 are:

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.162753654681801</td>
</tr>
<tr>
<td>1</td>
<td>0.162469864309526</td>
</tr>
<tr>
<td>2</td>
<td>0.162187553298139</td>
</tr>
<tr>
<td>3</td>
<td>0.16190670839297</td>
</tr>
<tr>
<td>4</td>
<td>0.161627318279375</td>
</tr>
<tr>
<td>5</td>
<td>0.161349369117073</td>
</tr>
<tr>
<td>6</td>
<td>0.16107284900106</td>
</tr>
<tr>
<td>7</td>
<td>0.160797745727675</td>
</tr>
<tr>
<td>8</td>
<td>0.160524047238664</td>
</tr>
<tr>
<td>9</td>
<td>0.160251741618955</td>
</tr>
<tr>
<td>10</td>
<td>0.159980817094486</td>
</tr>
<tr>
<td>11</td>
<td>0.160251741618955</td>
</tr>
</tbody>
</table>

---

824 • CrossCorrelation

IMSL C# Numerical Library
AutoCovariances of X are:

0 0
1 0.160524047238664
2 0.1607745727675
3 0.16107284900106
4 0.16134369117073
5 0.161627318279375
6 0.161906708839297
7 0.162187553298139
8 0.162469864309526
9 0.162753654681801
10 0.163040099734842
11 0.163328069921397
12 0.163618547762893
13 0.163911110444302
14 0.16420562443664
15 0.16450187815623
16 0.16480004153596
17 0.16510035412774
18 0.165402919471973
19 0.165707862229453
20 0.166014817514075

AutoCovariances of Y are:

0 0
1 0.2189370662893
2 0.92010118439122
3 9.15657243817617
4 8.0990196442277
5 6.94850770962479
6 5.87055032023953
7 4.9607624327211
8 4.25189969596136
9 3.73611877936647
10 3.37615547771905
11 3.13231605775447
12 2.89042463361545
13 2.68042845999682
14 2.50426970727032
15 2.3547186903426
16 2.23944263361545
17 2.14593790165038
18 2.067243817617
19 2.0086156
20 0.1951444

AutoCorrelations of X are:

0 0
1 0.952474916541448
2 0.834092130980584
3 0.81859776674248
4 0.531232576318765
5 0.407502117801518
6 0.31820082733867
7 0.26019453215175
8 0.227512619143721
9 0.213066602752258
10 0.208330776250152

AutoCorrelations of Y are:

0 0
1 0.970756656983059
2 0.970756656983059

Time Series and Forecasting
CrossCorrelation.StdErr Enumeration

public enumeration Imsl.Stat.CrossCorrelation.StdErr

Standard Error computation method.

Fields

Bartletts

Description
Indicates standard error computation using Bartlett’s formula.

BartlettsNoCC
public Imsl.Stat.CrossCorrelation.StdErr BartlettsNoCC

Description
Indicates standard error computation using Bartlett’s formula with the assumption of no cross-correlation.

MultiCrossCorrelation Class

public class Imsl.Stat.MultiCrossCorrelation

Computes the multichannel cross-correlation function of two mutually stationary multichannel time series.
MultiCrossCorrelation estimates the multichannel cross-correlation function of two mutually stationary multichannel time series. Define the multichannel time series \( X \) by

\[
X = (X_1, X_2, \ldots, X_p)
\]

where

\[
X_j = (X_{1j}, X_{2j}, \ldots, X_{nj})^T, \quad j = 1, 2, \ldots, p
\]

with \( n = X.GetLength(0) \) and \( p = X.GetLength(1) \). Similarly, define the multichannel time series \( Y \) by

\[
Y = (Y_1, Y_2, \ldots, Y_q)
\]

where

\[
Y_j = (Y_{1j}, Y_{2j}, \ldots, Y_{mj})^T, \quad j = 1, 2, \ldots, q
\]

with \( m = Y.GetLength(0) \) and \( q = Y.GetLength(1) \). The columns of \( X \) and \( Y \) correspond to individual channels of multichannel time series and may be examined from a univariate perspective. The rows of \( X \) and \( Y \) correspond to observations of \( p \)-variate and \( q \)-variate time series, respectively, and may be examined from a multivariate perspective. Note that an alternative characterization of a multivariate time series \( X \) considers the columns to be observations of the multivariate time series while the rows contain univariate time series. For example, see Priestley (1981, page 692) and Fuller (1976, page 14).

Let \( \hat{\mu}_X = x\text{mean} \) be the row vector containing the means of the channels of \( X \). In particular,

\[
\hat{\mu}_X = (\hat{\mu}_{X_1}, \hat{\mu}_{X_2}, \ldots, \hat{\mu}_{X_p})
\]

where for \( j = 1, 2, \ldots, p \)

\[
\hat{\mu}_{X_j} = \begin{cases} 
\mu_{X_j} & \text{for } \mu_{X_j} \text{ known} \\
\frac{1}{n} \sum_{i=1}^{n} X_{ij} & \text{for } \mu_{X_j} \text{ unknown}
\end{cases}
\]

Let \( \hat{\mu}_Y = y\text{mean} \) be similarly defined. The cross-covariance of lag \( k \) between channel \( i \) of \( X \) and channel \( j \) of \( Y \) is estimated by

\[
\hat{\sigma}_{X_i Y_j}(k) = \begin{cases} 
\frac{1}{N} \sum_{i=1}^{n} (X_{i}-\hat{\mu}_{X_i})(Y_{i+k,j}-\hat{\mu}_{Y_j}) & k = 0, 1, \ldots, K \\
\frac{1}{N} \sum_{i=1}^{n} (X_{i}-\hat{\mu}_{X_i})(Y_{i+k,j}-\hat{\mu}_{Y_j}) & k = -1, -2, \ldots, -K
\end{cases}
\]

where \( i = 1, \ldots, p, j = 1, \ldots, q, \) and \( K = \text{maxLag} \). The summation on \( i \) extends over all possible cross-products with \( N \) equal to the number of cross-products in the sum.

Let \( \hat{\sigma}_X(0) = x\text{var} \), where \( x\text{var} \) is the variance of \( X \), be the row vector consisting of estimated variances of the channels of \( X \). In particular,

\[
\hat{\sigma}_X(0) = (\hat{\sigma}_{X_1}(0), \hat{\sigma}_{X_2}(0), \ldots, \hat{\sigma}_{X_p}(0))
\]

where

\[
\hat{\sigma}_{X_j}(0) = \frac{1}{n} \sum_{i=1}^{n} (X_{ij} - \hat{\mu}_{X_j})^2, \quad j = 0, 1, \ldots, p
\]

Let \( \hat{\sigma}_Y(0) = y\text{var} \), where \( y\text{var} \) is the variance of \( Y \), be similarly defined. The cross-correlation of lag \( k \) between channel \( i \) of \( X \) and channel \( j \) of \( Y \) is estimated by

\[
\hat{\rho}_{X_j Y_j}(k) = \frac{\hat{\sigma}_{X_j Y_j}(k)}{[\hat{\sigma}_X(0)\hat{\sigma}_Y(0)]^{\frac{1}{2}}} \quad k = 0, \pm 1, \ldots, \pm K
\]
**Property**

**NumberOfProcessors**

```csharp
public int NumberOfProcessors {get; set; }
```

**Description**

Perform the parallel calculations with the maximum possible number of processors set to `NumberOfProcessors`.

**Property Value**

An int indicating the maximum possible number of processors to use.

**Remarks**

By default, `NumberOfProcessors = Environment.ProcessorCount`. If `NumberOfProcessors` is set to a number less than 1 or greater than `Environment.ProcessorCount`, `Environment.ProcessorCount` will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**Constructor**

**MultiCrossCorrelation**

```csharp
public MultiCrossCorrelation(double[,] x, double[,] y, int maximumLag)
```

**Description**

Constructor to compute the multichannel cross-correlation function of two mutually stationary multichannel time series.

**Parameters**

- `x` – A two-dimensional `double` array containing the first multichannel stationary time series. Each row of `x` corresponds to an observation of a multivariate time series and each column of `x` corresponds to a univariate time series.
- `y` – A two-dimensional `double` array containing the second multichannel stationary time series. Each row of `y` corresponds to an observation of a multivariate time series and each column of `y` corresponds to a univariate time series.
- `maximumLag` – A int containing the maximum lag of the cross-covariance and cross-correlations to be computed. `maximumLag` must be greater than or equal to 1 and less than the minimum number of observations of `x` and `y`.

**Methods**

**GetCrossCorrelation**

```csharp
public double[,] GetCrossCorrelation()
```
Returns the cross-correlations between the channels of x and y.

Returns
A double array of size 2 * maximumLag +1 by x.GetLength(1) by y.GetLength(1) containing the cross-correlations between the time series x and y.

Remarks
The cross-correlation between channel i of the x series and channel j of the y series at lag k, where k = -maximumLag, ..., 0, 1, ..., maximumLag, corresponds to output array CC[k,i,j] where k= 0, 1, ..., (2*maximumLag), i = 1, ..., x.GetLength(1), and j = 1, ..., y.GetLength(1).

Exception
Imsl.Stat.NonPosVarianceXYException is thrown if the problem is ill-conditioned. The variance is too small to work with.

GetCrossCovariance
public double[,,] GetCrossCovariance()

Description
Returns the cross-covariances between the channels of x and y.

Returns
A double array of size 2 * maximumLag +1 by x.GetLength(1) by y.GetLength(1) containing the cross-covariances between the time series x and y.

Remarks
The cross-covariances between channel i of the x series and channel j of the y series at lag k where k = -maximumLag, ..., 0, 1, ..., maximumLag, corresponds to output array CCV[k,i,j] where k= 0, 1, ..., (2*maximumLag), i = 1, ..., x.GetLength(1), and j = 1, ..., y.GetLength(1).

Exception
Imsl.Stat.NonPosVarianceXYException is thrown if the problem is ill-conditioned. The variance is too small to work with.

GetMeanX
public double[] GetMeanX()

Description
Returns an estimate of the mean of each channel of x.

Returns
A one-dimensional double containing the estimate of the mean of each channel in time series x.

GetMeanY
public double[] GetMeanY()

Description
Returns an estimate of the mean of each channel of y.
Returns
A one-dimensional double containing the estimate of the mean of each channel in the time series y.

GetVarianceX
public double[] GetVarianceX()

Description
Returns the variances of the channels of x.

Returns
A one-dimensional double containing the variances of each channel in the time series x.

Exception
Imsl.Stat.NonPosVarianceXYException is thrown if the problem is ill-conditioned. The variance is too small to work with.

GetVarianceY
public double[] GetVarianceY()

Description
Returns the variances of the channels of y.

Returns
A one-dimensional double containing the variances of each channel in the time series y.

Exception
Imsl.Stat.NonPosVarianceXYException is thrown if the problem is ill-conditioned. The variance is too small to work with.

Example 1: MultiCrossCorrelation
Consider the Wolfer Sunspot Data (Y) (Box and Jenkins 1976, page 530) along with data on northern light activity (X1) and earthquake activity (X2) (Robinson 1967, page 204) to be a three-channel time series. Methods GetCrossCovariance and GetCrossCorrelation are used to compute the cross-covariances and cross-correlations between X1 and Y and between X2 and Y with lags from $-\text{maximumLag} = -10$ through lag $\text{maximumLag} = 10$.

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;
using Matrix = Imsl.Math.Matrix;

public class MultiCrossCorrelationEx1
{
    public static void Main(String[] args)
    {
        int i;
        double[,] x = {
            {155.0, 66.0}, {113.0, 62.0},
            {3.0, 66.0}, {10.0, 197.0},
```
double[,] y = {{101.0}, {82.0},
{66.0}, {35.0},
{31.0}, {7.0},
{20.0}, {92.0},
{154.0}, {126.0},
{85.0}, {68.0},
{38.0}, {23.0},
{0.0, 63.0}, {0.0, 0.0},
{12.0, 121.0}, {86.0, 0.0},
{102.0, 113.0}, {20.0, 27.0},
{98.0, 107.0}, {116.0, 50.0},
{187.0, 122.0}, {131.0, 127.0},
{168.0, 152.0}, {173.0, 216.0},
{238.0, 171.0}, {146.0, 70.0},
{0.0, 141.0}, {0.0, 69.0},
{0.0, 160.0}, {0.0, 92.0},
{12.0, 70.0}, {0.0, 46.0},
{37.0, 96.0}, {14.0, 78.0},
{11.0, 110.0}, {28.0, 79.0},
{19.0, 85.0}, {30.0, 113.0},
{11.0, 59.0}, {26.0, 86.0},
{0.0, 199.0}, {29.0, 53.0},
{47.0, 81.0}, {36.0, 81.0},
{35.0, 156.0}, {17.0, 27.0},
{0.0, 81.0}, {3.0, 107.0},
{6.0, 152.0}, {18.0, 99.0},
{15.0, 177.0}, {0.0, 48.0},
{3.0, 70.0}, {9.0, 158.0},
{64.0, 22.0}, {126.0, 43.0},
{38.0, 102.0}, {33.0, 111.0},
{71.0, 90.0}, {24.0, 86.0},
{20.0, 119.0}, {22.0, 82.0},
{13.0, 79.0}, {35.0, 111.0},
{84.0, 60.0}, {119.0, 118.0},
{86.0, 206.0}, {71.0, 122.0},
{115.0, 134.0}, {91.0, 131.0},
{43.0, 84.0}, {67.0, 100.0},
{60.0, 99.0}, {49.0, 99.0},
{100.0, 69.0}, {150.0, 67.0},
{178.0, 26.0}, {187.0, 106.0},
{76.0, 108.0}, {75.0, 155.0},
{100.0, 40.0}, {68.0, 75.0},
{93.0, 99.0}, {20.0, 86.0},
{51.0, 127.0}, {72.0, 201.0},
{118.0, 76.0}, {146.0, 64.0},
{101.0, 31.0}, {61.0, 138.0},
{87.0, 163.0}, {53.0, 98.0},
{69.0, 70.0}, {46.0, 155.0},
{47.0, 97.0}, {35.0, 82.0},
{74.0, 90.0}, {104.0, 122.0},
{97.0, 70.0}, {106.0, 96.0},
{113.0, 111.0}, {103.0, 42.0},
{68.0, 97.0}, {67.0, 91.0},
{82.0, 64.0}, {89.0, 81.0},
{102.0, 162.0}, {110.0, 137.0}};
MultiCrossCorrelation mcc =
    new MultiCrossCorrelation(x, y, 10);

new PrintMatrix("Mean of X : ").Print(mcc.GetMeanX());
new PrintMatrix("Variance of X : ").Print(mcc.GetVarianceX());
new PrintMatrix("Mean of Y : ").Print(mcc.GetMeanY());
new PrintMatrix("Variance of Y : ").Print(mcc.GetVarianceY());

double[,] tmpArr = new double[x.GetLength(1), y.GetLength(1)];
double[,] ccv = mcc.GetCrossCovariance();
Console.Out.WriteLine
    ("Multichannel cross-covariance between X and Y");
for (i = 0; i < 21; i++)
{
    for (int j=0; j<x.GetLength(1); j++)
        for (int k=0; k<y.GetLength(1); k++)
            tmpArr[j,k] = ccv[i,j,k];
    Console.Out.WriteLine("Lag K = " + (i - 10));
    new PrintMatrix("CrossCovariances :").Print(tmpArr);
}

double[,,] cc = mcc.GetCrossCorrelation();
Console.Out.WriteLine
("Multichannel cross-correlation between X and Y");
for (i = 0; i < 21; i++)
{
    for (int j=0; j<x.GetLength(1); j++)
        for (int k=0; k<y.GetLength(1); k++)
            tmpArr[j,k] = cc[i,j,k];
    Console.Out.WriteLine("Lag K = " + (i - 10));
    new PrintMatrix("CrossCorrelations :").Print(tmpArr);
}

Output

Mean of X :

0
0 63.43
1 97.97

Variance of X :

0
0 2643.6851
1 1978.4291

Mean of Y :

0
0 46.94

Variance of Y :

0
0 1383.7564

Multichannel cross-covariance between X and Y
Lag K = -10
CrossCovariances :

0
0 -20.5123555555557
1 70.7132444444444

Lag K = -9
CrossCovariances :

0
0 65.0243098901099
1 38.1363054945055

Time Series and Forecasting  MultiCrossCorrelation • 833
Lag K = -8
CrossCovariances:
0 0 216.637243478261
1 135.57832173913

Lag K = -7
CrossCovariances:
0 0 246.793769892473
1 100.362230107527

Lag K = -6
CrossCovariances:
0 0 142.127923404255
1 44.9678638297872

Lag K = -5
CrossCovariances:
0 0 50.6970421052632
1 -11.8094631578948

Lag K = -4
CrossCovariances:
0 0 72.6846166666667
1 32.6926333333334

Lag K = -3
CrossCovariances:
0 0 217.854096907217
1 -40.1185092783505

Lag K = -2
CrossCovariances:
0 0 355.820628571429
1 -152.649118367347

Lag K = -1
CrossCovariances:
0 0 579.653492929293
1 -212.95022020202

Lag K = 0
CrossCovariances:
0 0 821.6258
1 -104.7518

Lag K = 1
CrossCovariances :
0
0  810.131371717171
1  55.1601838383839

Lag K = 2
CrossCovariances :
0
0  628.385118367347
1  84.7751673469388

Lag K = 3
CrossCovariances :
0
0  438.271931958763
1  75.9630371134021

Lag K = 4
CrossCovariances :
0
0  238.792741666667
1  200.383466666667

Lag K = 5
CrossCovariances :
0
0  143.621147368421
1  282.986431578947

Lag K = 6
CrossCovariances :
0
0  252.973774468085
1  234.393289361702

Lag K = 7
CrossCovariances :
0
0  479.468286021505
1  223.033735483871

Lag K = 8
CrossCovariances :
0
0  724.912243478261
1  124.456582608696

Lag K = 9
CrossCovariances :
0
0  924.971232967034
1  -79.5174307692309

Lag K = 10
CrossCovariances :
0

Time Series and Forecasting
Multichannel cross-correlation between X and Y

<table>
<thead>
<tr>
<th>Lag K</th>
<th>CrossCorrelations</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-0.0107245938219684</td>
</tr>
<tr>
<td></td>
<td>0.0427376557935899</td>
</tr>
<tr>
<td>-9</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.0339970370656115</td>
</tr>
<tr>
<td></td>
<td>0.023048812287829</td>
</tr>
<tr>
<td>-8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.113265706453004</td>
</tr>
<tr>
<td></td>
<td>0.0819407975561327</td>
</tr>
<tr>
<td>-7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.129032618058936</td>
</tr>
<tr>
<td></td>
<td>0.0606569035081169</td>
</tr>
<tr>
<td>-6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.074309566502109</td>
</tr>
<tr>
<td></td>
<td>0.0271776680766982</td>
</tr>
<tr>
<td>-5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.0265062285548632</td>
</tr>
<tr>
<td></td>
<td>-0.00713740085770933</td>
</tr>
<tr>
<td>-4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.0380021196855836</td>
</tr>
<tr>
<td></td>
<td>0.0197587668528454</td>
</tr>
<tr>
<td>-3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.11390192098873</td>
</tr>
<tr>
<td></td>
<td>-0.0242468161934945</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.186035762912295</td>
</tr>
<tr>
<td>Lag K = -1</td>
<td>CrossCorrelations :</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>0</td>
<td>0.30363597562697</td>
</tr>
<tr>
<td>1</td>
<td>-0.128702809263875</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lag K = 0</th>
<th>CrossCorrelations :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.429575382251174</td>
</tr>
<tr>
<td>1</td>
<td>-0.0633098708358119</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lag K = 1</th>
<th>CrossCorrelations :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.423565683647071</td>
</tr>
<tr>
<td>1</td>
<td>0.0333377002981115</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lag K = 2</th>
<th>CrossCorrelations :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.328542235922487</td>
</tr>
<tr>
<td>1</td>
<td>0.051236397797642</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lag K = 3</th>
<th>CrossCorrelations :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.22914425606054</td>
</tr>
<tr>
<td>1</td>
<td>0.0459105243818767</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lag K = 4</th>
<th>CrossCorrelations :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.124849394067548</td>
</tr>
<tr>
<td>1</td>
<td>0.121107717407232</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lag K = 5</th>
<th>CrossCorrelations :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.075090277447643</td>
</tr>
<tr>
<td>1</td>
<td>0.171031279954621</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lag K = 6</th>
<th>CrossCorrelations :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.132263745693782</td>
</tr>
<tr>
<td>1</td>
<td>0.141662566889261</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lag K = 7</th>
<th>CrossCorrelations :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.250683184784367</td>
</tr>
<tr>
<td>1</td>
<td>0.134797082107539</td>
</tr>
</tbody>
</table>

**Time Series and Forecasting**

**MultiCrossCorrelation** • 837
Lag K = 8
CrossCorrelations :
 0 0.37901007257894
 1 0.0752190432013873

Lag K = 9
CrossCorrelations :
 0 0.48360807434863
 1 -0.0480587280714567

Lag K = 10
CrossCorrelations :
 0 0.48245160241607
 1 -0.168795069078383

### LackOfFit Class

```csharp
public class Imsl.Stat.LackOfFit
```

Performs lack-of-fit test for a univariate time series or transfer function given the appropriate correlation function.

LackOfFit may be used to diagnose lack of fit in both ARMA and transfer function models. Typical arguments for these situations are:

<table>
<thead>
<tr>
<th>Model</th>
<th>lagMin</th>
<th>lagMax</th>
<th>npFree</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMA ( (p, q) )</td>
<td>1</td>
<td>( \sqrt{n\text{Observations}} )</td>
<td>( p + q )</td>
</tr>
<tr>
<td>Transfer function</td>
<td>0</td>
<td>( \sqrt{n\text{Observations}} )</td>
<td>( r + s )</td>
</tr>
</tbody>
</table>

LackOfFit performs a portmanteau lack of fit test for a time series or transfer function containing \( n\text{Observations} \) observations given the appropriate sample correlation function \( \hat{\rho}(k) \) for \( k = L, L+1, \ldots, K \) where \( L = \text{lagMin} \) and \( K = \text{lagMax} \).

The basic form of the test statistic \( Q \) is

\[
Q = n(n + 2) \sum_{k=L}^{K} (n-k)^{-1} \hat{\rho}(k)
\]

with \( L = 1 \) if \( \hat{\rho}(k) \) is an autocorrelation function. Given that the model is adequate, \( Q \) has a chi-squared distribution with \( K - L + 1 - m \) degrees of freedom where \( m = \text{npFree} \) is the number of parameters estimated in the model. If the mean of the time series is estimated, Woodfield (1990) recommends not including this in the count of the parameters estimated in the model. Thus, for an ARMA\((p, q)\) model set
\[ npFree = p + q \] regardless of whether the mean is estimated or not. The original derivation for time series models is due to Box and Pierce (1970) with the above modified version discussed by Ljung and Box (1978). The extension of the test to transfer function models is discussed by Box and Jenkins (1976, pages 394-395).

**Methods**

**Compute**

```java
static public double[] Compute(int nObservations, double[] correlations, int npFree, int lagMax)
```

**Description**

Performs lack-of-fit test for a univariate time series or transfer function given the appropriate correlation function using a minimum lag of 1.

**Parameters**

- `nObservations` – An int containing the number of observations of the stationary time series.
- `correlations` – A double array of length `lagMax+1` containing the correlation function.
- `npFree` – An int scalar specifying the number of free parameters in the formulation of the time series model. `npFree` must be greater than or equal to zero and less than `lagMax`. Woodfield (1990) recommends `npFree = p + q`.
- `lagMax` – An int scalar specifying the maximum lag of the correlation function.

**Returns**

A double array of length 2 with the test statistic, \( Q \), and its \( p \)-value, \( p \). Under the null hypothesis, \( Q \) has an approximate chi-squared distribution with \( \text{lagMax-lagMin+1-npFree} \) degrees of freedom.

```java
static public double[] Compute(int nObservations, double[] correlations, int npFree, int lagMax, int lagMin)
```

**Description**

Performs lack-of-fit test for a univariate time series or transfer function given the appropriate correlation function.

**Parameters**

- `nObservations` – An int containing the number of observations of the stationary time series.
- `correlations` – A double array of length `lagMax+1` containing the correlation function.
- `npFree` – An int scalar specifying the number of free parameters in the formulation of the time series model. `npFree` must be greater than or equal to zero and less than `lagMax`. Woodfield (1990) recommends `npFree = p + q`.
- `lagMax` – An int scalar specifying the maximum lag of the correlation function.
lagMin – An int scalar specifying the minimum lag of the correlation function. lagMin corresponds to the lower bound of summation in the lack of fit test statistic.

Default: lagMin = 1.

Returns
A double array of length 2 with the test statistic, Q, and its p-value, p. Under the null hypothesis, Q has an approximate chi-squared distribution with lagMax-lagMin+1-npFree degrees of freedom.

Example: Lack Of Fit

Consider the Wolfer Sunspot Data (Anderson 1971, page 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869. An ARMA(2,1) with nonzero mean is fitted using the ARMA class. The autocorrelations of the residuals are estimated using the AutoCorrelation class. Class LackOfFit is used to compute a portmanteau lack of fit test using 10 lags. The warning messages from ARMA in the output can be ignored. (See Example 2 in ARMA for a full explanation of the warning messages.)

```csharp
using System;
using Imsl.Stat;

class LackOfFitEx1
{
    public static void Main(String[] args)
    {
        int lagmax = 10;
        int npfree = 4;
        double tolerance = 0.125;

        // sunspot data for 1770 through 1869
        double[] x = { 100.8, 81.6, 66.5, 34.8, 30.6, 7, 19.8,
                        92.5, 154.4, 125.9, 84.8, 68.1, 38.5, 22.8, 10.2, 24.1, 82.9,
                        132, 130.9, 118.1, 89.9, 66.6, 60, 46.9, 41, 21.3, 16, 6.4,
                        4.1, 6.8, 14.5, 34, 45, 43.1, 47.5, 42.2, 28.1, 10.1, 8.1,
                        2.5, 0, 1.4, 5, 12.2, 13.9, 35.4, 45.8, 41.1, 30.4, 23.9,
                        15.7, 6.6, 4, 1.8, 8.5, 16.6, 36.3, 49.7, 62.5, 67, 71, 47.8,
                        27.5, 8.5, 13.2, 56.9, 121.5, 138.3, 103.2, 85.8, 63.2, 36.8,
                        24.2, 10.7, 15, 40.1, 61.5, 98.5, 124.3, 95.9, 66.5, 64.5,
                        54.2, 39, 20.6, 6.7, 4.3, 22.8, 54.8, 93.8, 95.7, 77.2, 59.1,
                        44, 47, 30.5, 16.3, 7.3, 37.3, 73.9};

        // Get residuals from ARMA(2,1) for autocorrelation/lack of fit
        ARMA arma = new ARMA(2, 1, x);
        arma.Method = ARMA.ParamEstimation.LeastSquares;
        arma.ConvergenceTolerance = tolerance;
        arma.Compute();
        double[] residuals = arma.GetResidual();

        // Get autocorrelations from residuals for lack of fit test
        AutoCorrelation ac = new AutoCorrelation(residuals, lagmax);
        double[] correlations = ac.GetAutoCorrelations();
    }
}
```
// Get lack of fit test statistic and p-value
  double[] result = LackOfFit.Compute(x.Length, correlations, npfree, lagmax);
  // Print parameter estimates, test statistic, and p-value
  // NOTE: Test Statistic Q follows a Chi-squared dist.
  Console.WriteLine("Lack of Fit Statistic, Q = " + result[0]);
  Console.WriteLine("P-value of Q = " + result[1]);
}\}

**Output**

Lack of Fit Statistic, Q = 14.6060180633065
P-value of Q = 0.97644740013821

Imsl.Stat.ARMA: Relative function convergence - Both the scaled actual and predicted reductions in the function are less than or equal to the relative function convergence tolerance "ConvergenceTolerance" = 0.0645856533065147.

Imsl.Stat.ARMA: Least squares estimation of the parameters has failed to converge. Increase "length" and/or "tolerance" and/or "convergence_tolerance". The estimates of the parameters at the last iteration may be used as new starting values.

---

**ARAutoUnivariate Class**

```java
public class Imsl.Stat.ARAutoUnivariate
{
    // Automatically determines the best autoregressive time series model using Akaike's Information Criterion.
    Automatic.Univariate.ARAutoUnivariate
    automatically selects the order of the AR model that best fits the data and then computes the AR coefficients. The algorithm used in ARAutoUnivariate is derived from the work of Akaike, H., et. al (1979) and Kitagawa and Akaike (1978). This code was adapted from the UNIMAR procedure published as part of the TIMSAC-78 Library.

    The best fit AR model is determined by successively fitting AR models with 0, 1, 2, ..., maxlag autoregressive coefficients. For each model, Akaike's Information Criterion (AIC) is calculated based on the formula

    \[
    \text{AIC} = -2 \ln(\text{likelihood}) + 2p
    \]

    Class ARAutoUnivariate uses the approximation to this formula developed by Ozaki and Oda (1979),

    \[
    \text{AIC} \approx (n - \text{maxlag}) \ln(\hat{\sigma}^2) + 2(p + 1) + (n - \text{maxlag})(\ln(2\pi) + 1)
    \]

    where \(\hat{\sigma}^2\) is an estimate of the residual variance of the series, commonly known in time series analysis as the innovation variance and \(n\) is the number of observations in the time series \(z\), \(n = z\cdot\text{Length}\). By dropping the constant

    \((n - \text{maxlag})(\ln(2\pi) + 1),\)
```
the calculation is simplified to

$$AIC \approx (n - \text{maxlag}) \ln(\hat{\sigma}^2) + 2(p + 1),$$

The best fit model is the model with minimum AIC. If the number of parameters in this model selected by ARAutoUnivariate is equal to the highest order autoregressive model fitted, i.e., \(p=\text{maxlag}\), then a model with smaller AIC might exist for larger values of \text{maxlag}. In this case, increasing \text{maxlag} to explore AR models with additional autoregressive parameters might be warranted.

Property EstimationMethod can be used to specify the method used to calculate the AR coefficients. If EstimationMethod is set to MethodOfMoments, estimates of the autoregressive coefficients for the model with minimum AIC are calculated using method of moments as described in the ARMA class. If LeastSquares is specified, the coefficients are determined by the method of least squares applied in the form described by Kitagawa and Akaike (1978). If MaximumLikelihood is specified, the coefficients are estimated using maximum likelihood as described in the ARMA\text{MaxLikelihood} class.

### Properties

**AIC**

```csharp
public double AIC { get; }
```

**Description**

The final estimate for Akaike’s Information Criterion (AIC) at the optimum.

**Property Value**

A double scalar value which is an approximation to 

$$AIC = -2\ln(L) + 2p,$$

where \(L\) is the value of the maximum likelihood function evaluated at the parameter estimates. The approximation uses the calculation

$$AIC \approx (n - \text{maxlag}) \ln(\hat{\sigma}^2) + 2(p + 1) + (n - \text{maxlag})(\ln(2\pi) + 1),$$

where \(\hat{\sigma}^2\) is an estimate of the residual variance of the series, commonly known in time series analysis as the innovation variance, and \(n\) is the number of observations in the time series \(z (n=z.Length)\).

**Exceptions**

- `Imsl.Stat.MatrixSingularException` is thrown if the input matrix is singular.
- `Imsl.Stat.TooManyCallsException` is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- `Imsl.Stat.IncreaseErrRelException` is thrown if the bound for the relative error is too small.
- `Imsl.Stat.NewInitialGuessException` is thrown if the iteration has not made good progress.
- `Imsl.Stat.IllConditionedException` is thrown if the problem is ill-conditioned.
- `Imsl.Stat.TooManyIterationsException` is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.

Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.

Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.

Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.

---

**BackwardOrigin**

```csharp
public int BackwardOrigin { get; set; }
```

**Description**

The maximum backward origin used in calculating the forecasts.

**Property Value**

An int scalar containing the maximum backward origin.

**Remarks**

BackwardOrigin must be greater than or equal to 0 and less than or equal to \( z.\text{Length} - p \), where \( p \) is the optimum number of AR coefficients determined by ARAutoUnivariate. The forecasted values returned will be \( z.\text{Length} - \text{BackwardOrigin} \) through \( z.\text{Length} \). By default, BackwardOrigin = 0.

---

**Confidence**

```csharp
public double Confidence { get; set; }
```

**Description**

The confidence level for calculating confidence limit deviations returned from GetDeviations.

**Property Value**

A double scalar value representing the confidence level used in computing forecast confidence intervals.

**Remarks**

Typical choices for Confidence are 0.90, 0.95, and 0.99. Confidence must be greater than 0.0 and less than 1.0. By default, Confidence = 0.95.

---

**Constant**

```csharp
public double Constant { get; }
```

**Description**

The estimate for the constant parameter in the ARMA series.

**Property Value**

A double scalar equal to the estimate for the constant parameter in the ARMA series.
Exceptions

Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.

ConvergenceTolerance

public double ConvergenceTolerance {get; set; }

Description

The tolerance level used to determine convergence of the nonlinear least-squares and maximum likelihood algorithms.

Property Value

A double scalar containing the tolerance level used to determine convergence of the nonlinear least-squares and maximum likelihood algorithms. ConvergenceTolerance represents the minimum relative decrease in sum of squares between two iterations required to determine convergence.

Remarks

ConvergenceTolerance must be greater than or equal to 0. By default, ConvergenceTolerance = 10^{-10}.

EstimationMethod

public Imsl.Stat.ARAutoUnivariate.ParamEstimation EstimationMethod {get; set; }

Description

The estimation method used for estimating the final estimates for the autoregressive coefficients.

Property Value

A ParamEstimation specifying the method used to estimate the autoregressive parameters estimates.
### Remarks

<table>
<thead>
<tr>
<th></th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MethodOfMoments</td>
<td>Autoregressive parameters are estimated by a method of moments procedure.</td>
</tr>
<tr>
<td>LeastSquares</td>
<td>Autoregressive parameters are estimated by a least-squares procedure.</td>
</tr>
<tr>
<td>MaxLikelihood</td>
<td>Autoregressive parameters are estimated by a maximum likelihood procedure.</td>
</tr>
</tbody>
</table>


### InnovationVariance

**public double InnovationVariance {get; }**

**Description**
The final estimate for the innovation variance.

**Property Value**
A double scalar value equal to the estimate for the innovation variance.

**Exceptions**
- Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
- Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
- Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
- Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
- Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
- Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
- Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
- Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.
- Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.

### Likelihood

**public double Likelihood {get; }**
Description

The final estimate for \( L \approx e^{-(\text{AIC} - 2p)/2} \), where \( p \) is the AR order, AIC is the value of Akaike’s Information Criterion, and \( L \) is the likelihood function evaluated for the optimum autoregressive model.

Property Value

A double scalar equal to the estimate for the \( L \approx e^{-(\text{AIC} - 2p)/2} \).

Remarks

If MaximumLikelihood is chosen as the EstimationMethod, the exact likelihood evaluated for the optimum autoregressive model will be returned instead. Otherwise it is calculated using the approximation to the AIC.

Exceptions

- `Imsl.Stat.MatrixSingularException` is thrown if the input matrix is singular.
- `Imsl.Stat.TooManyCallsException` is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- `Imsl.Stat.IncreaseErrRelException` is thrown if the bound for the relative error is too small.
- `Imsl.Stat.NewInitialGuessException` is thrown if the iteration has not made good progress.
- `Imsl.Stat.IllConditionedException` is thrown if the problem is ill-conditioned.
- `Imsl.Stat.TooManyIterationsException` is thrown if the maximum number of iterations is exceeded.
- `Imsl.Stat.TooManyFunctionEvaluationsException` is thrown if the maximum number of function evaluations is exceeded.
- `Imsl.Stat.TooManyJacobianEvalException` is thrown if the maximum number of Jacobian evaluations is exceeded.
- `Imsl.Stat.SingularTriangularMatrixException` is thrown if the input triangular matrix is singular.
- `Imsl.Stat.NonStationaryException` is thrown if the final maximum likelihood estimates for the time series are nonstationary.

MaxIterations

```csharp
public int MaxIterations {get; set; }
```

Description

The maximum number of iterations used for estimating the autoregressive coefficients.

Property Value

An int scalar containing the maximum number of iterations allowed in the nonlinear equation solver used in both the method of moments and least-squares algorithms.
Remarks
MaxIterations must be greater than 0. By default, MaxIterations = 300.

Maxlag
public int Maxlag {get; }

Description
The current value used to represent the maximum number of autoregressive lags to achieve the minimum AIC.

Property Value
An int containing the maximum number of autoregressive lags evaluated.

Mean
public double Mean {get; set; }

Description
The mean used to center the time series \( z \).

Property Value
A double containing the value used to center the series before searching for the optimum number of AR lags. This is equal to the mean of the time series \( z \) unless the Mean is set.

Remarks
By default, the time series \( z \) is centered about its sample mean.

Exceptions
Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.
**NumberOfProcessors**

```csharp
public int NumberOfProcessors {get; set; }
```

**Description**
Perform the parallel calculations with the maximum possible number of processors set to `NumberOfProcessors`.

**Property Value**
An `int` indicating the maximum possible number of processors to use.

**Remarks**
By default, `NumberOfProcessors = Environment.ProcessorCount`. If `NumberOfProcessors` is set to a number less than 1 or greater than `Environment.ProcessorCount`, `Environment.ProcessorCount` will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**Order**

```csharp
public int Order {get; }
```

**Description**
The order of the AR model selected with the minimum AIC.

**Property Value**
An `int` containing the optimum AR order selected by `ARAutoUnivariate`'s minimum AIC selection.

**Exceptions**
- `Imsl.Stat.MatrixSingularException` is thrown if the input matrix is singular.
- `Imsl.Stat.TooManyCallsException` is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- `Imsl.Stat.IncreaseErrRelException` is thrown if the bound for the relative error is too small.
- `Imsl.Stat.NewInitialGuessException` is thrown if the iteration has not made good progress.
- `Imsl.Stat.IllConditionedException` is thrown if the problem is ill-conditioned.
- `Imsl.Stat.TooManyIterationsException` is thrown if the maximum number of iterations is exceeded.
- `Imsl.Stat.TooManyFunctionEvaluationsException` is thrown if the maximum number of function evaluations is exceeded.
- `Imsl.Stat.TooManyJacobianEvalException` is thrown if the maximum number of Jacobian evaluations is exceeded.
- `Imsl.Stat.SingularTriangularMatrixException` is thrown if the input triangular matrix is singular.
- `Imsl.Stat.NonStationaryException` is thrown if the final maximum likelihood estimates for the time series are nonstationary.

**TimsacConstant**

```csharp
public double TimsacConstant {get; }
```

---

848 • `ARAutoUnivariate` IMSL C# Numerical Library
Description
The estimate for the constant parameter in the ARMA series.

Property Value
A double scalar equal to the estimate for the constant parameter in the ARMA series.

Exceptions

- `Imsl.Stat.MatrixSingularException` is thrown if the input matrix is singular.
- `Imsl.Stat.TooManyCallsException` is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- `Imsl.Stat.IncreaseErrRelException` is thrown if the bound for the relative error is too small.
- `Imsl.Stat.NewInitialGuessException` is thrown if the iteration has not made good progress.
- `Imsl.Stat.IllConditionedException` is thrown if the problem is ill-conditioned.
- `Imsl.Stat.TooManyIterationsException` is thrown if the maximum number of iterations is exceeded.
- `Imsl.Stat.TooManyFunctionEvaluationsException` is thrown if the maximum number of function evaluations is exceeded.
- `Imsl.Stat.TooManyJacobianEvalException` is thrown if the maximum number of Jacobian evaluations is exceeded.
- `Imsl.Stat.SingularTriangularMatrixException` is thrown if the input triangular matrix is singular.
- `Imsl.Stat.NonStationaryException` is thrown if the final maximum likelihood estimates for the time series are nonstationary.

TimsacVariance
public double TimsacVariance {get; }  

Description
The final estimate for the innovation variance calculated by the TIMSAC automatic AR modeling routine (UNIMAR).

Property Value
A double scalar value equal to the estimate for the innovation variance.

Remarks
This variance depends upon the value of maxlag since the series length in this computation depends on its value. Use InnovationVariance to return the innovation variance calculated using the maximum series length.
Exceptions

Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.

Constructor

ARAutoUnivariate

public ARAutoUnivariate(int maxlag, double[] z)

Description
ARAutoUnivariate constructor.

Parameters

maxlag – An int scalar specifying the maximum number of autoregressive lags to evaluate.
z – A double array containing the time series.

Methods

Compute

public void Compute()
Description
Determines the autoregressive model with the minimum AIC by fitting autoregressive models from 0 to maxlag lags using the method of moments or an estimation method specified by the user through EstimationMethod.

Exceptions
  - Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
  - Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
  - Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
  - Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
  - Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
  - Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
  - Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
  - Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
  - Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.
  - Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.

Forecast
public double[,] Forecast(int nForecast)

Description
Returns forecasts and associated confidence interval offsets.

Parameter
  - nForecast – An int representing the number of requested forecasts.

Returns
A double matrix of dimension nForecast by BackwardOrigin + 1 containing the forecasts. The forecasts are for lead times l = 1, 2, ..., nForecast at origins z.Length-BackwardOrigin-1+j where j = 1, ..., BackwardOrigin+1.

Exceptions
  - Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
  - Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat_TOO_MANY_ITERATIONS is thrown if the maximum number of iterations is exceeded.
Imsl.Stat_TOO_MANY_FUNCTION_EVALUATIONS is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat_TOO_MANY_JACOBIAN_EVALS is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.

GetAR
public double[] GetAR()

Description
Returns the final autoregressive parameter estimates at the optimum AIC using the estimation method specified in EstimationMethod.

Returns
A double array containing the estimates for the autoregressive parameters.

Exceptions
Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat_TOO_MANY_CALLS is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat_TOO_MANY_ITERATIONS is thrown if the maximum number of iterations is exceeded.
Imsl.Stat_TOO_MANY_FUNCTION_EVALUATIONS is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat_TOO_MANY_JACOBIAN_EVALS is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.

GetDeviations

```java
public double[] GetDeviations()
```

**Description**

Returns the deviations for each forecast used for calculating the forecast confidence limits.

**Returns**

A double array of length BackwardOrigin+nForecast containing the deviations for calculating forecast confidence intervals. The confidence level is specified in the Confidence property.

**Remarks**

Note that the Forecast or GetForecast method must be invoked first before invoking this method. Otherwise, the method returns null.

GetForecast

```java
public double[] GetForecast(int nForecast)
```

**Description**

Returns a specified number of forecasts beyond the last value in the series.

**Parameter**

- `nForecast` – An int representing the number of requested forecasts beyond the last value in the series.

**Returns**

A double array containing the nForecast+BackwardOrigin forecasts. The first BackwardOrigin forecasts are one-step ahead forecasts for the last BackwardOrigin values in the series. The next nForecast values in the returned series are forecasts for the next values beyond the series.

**Exceptions**

- Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
- Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
- Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
- Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
- Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
- Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat_TOO_MANY_JACOBIAN_EVAL_EXCEPTION is thrown if the maximum number of Jacobian evaluations is exceeded.

Imsl.Stat.SINGULAR_TRIANGULAR_MATRIX_EXCEPTION is thrown if the input triangular matrix is singular.

Imsl.Stat.NON_STATIONARY_EXCEPTION is thrown if the final maximum likelihood estimates for the time series are nonstationary.

---

**GetResiduals**

```csharp
public double[] GetResiduals()
```

**Description**

Returns the current values of the vector of residuals.

**Returns**

A double array of length `BackwardOrigin` containing the residuals for the last `BackwardOrigin` values in the time series. This array will be null unless `Compute` and either the `Forecast` or `GetForecast` methods are called previously.

---

**GetTimeSeries**

```csharp
public double[] GetTimeSeries()
```

**Description**

Returns the time series used for estimating the minimum AIC and the autoregressive coefficients.

**Returns**

A double array containing the time series values set in the constructor.

---

**GetTimsacAR**

```csharp
public double[] GetTimsacAR()
```

**Description**

Returns the final auto regressive parameter estimates at the optimum AIC estimated by the original TIMSAC routine (UNIMAR).

**Returns**

A double array of length `p` containing the estimates for the autoregressive parameters.

**Remarks**

These estimates vary depending upon the value of `maxlag` since its value changes the length of the series used to estimate these parameters. Use the `GetAR` method to obtain the best estimates from the estimation method specified by `EstimationMethod` calculated using the maximum number of available observations.

**Exceptions**

- `Imsl.Stat.MatrixSingularException` is thrown if the input matrix is singular.
- `Imsl.Stat.TooManyCallsException` is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.

Example 1: ARAutoUnivariate

Consider the Wolfer Sunspot Data (Anderson 1971, p. 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869. In this example, ARAutoUnivariate found the minimum AIC fit is an autoregressive model with 3 lags:

\[ \tilde{W}_t = \phi_1 \tilde{W}_{t-1} + \phi_2 \tilde{W}_{t-2} + \phi_3 \tilde{W}_{t-3} + a_t, \]

where

\[ \tilde{W}_t := W_t - \mu \]

\( \mu \) is the sample mean of the time series \( \{W_t\} \). Defining the overall constant \( \theta_0 \) by \( \theta_0 := \mu(1 - \sum_{i=1}^{3} \phi_i) \), we obtain the following equivalent representation:

\[ W_t = \theta_0 + \phi_1 W_{t-1} + \phi_2 W_{t-2} + \phi_3 W_{t-3} + a_t. \]

The example computes estimates for \( \theta_0, \phi_1, \phi_2, \phi_3 \) for each of the three parameter estimation methods available.

using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class ARAutoUnivariateEx1
{
    public static void Main(String[] args)
    {
        double[] z = new double[] {100.8, 81.6, 66.5, 34.8, 30.6,
          7, 19.8, 92.5, 154.4, 125.9,
          84.8, 68.1, 38.5, 22.8, 10.2,
          24.1, 82.9, 132, 130.9, 118.1,
          89.9, 66.6, 60, 46.9, 41,
          Time Series and Forecasting
ARAutoUnivariate autoAR = new ARAutoUnivariate(20, z);
Console.Out.WriteLine(" Method of Moments ");
autoAR.EstimationMethod =
autoAR.Compute();

/* try another method */
Console.Out.WriteLine(" Least Squares ");
autoAR.EstimationMethod =
autoAR.Compute();

Console.Out.WriteLine(" Maximum Likelihood ");
autoAR.EstimationMethod =
autoAR.Compute();
Console.Out.WriteLine("Order Selected: " + autoAR.Order);
Console.Out.WriteLine("AIC = " + autoAR.AIC + " Variance = " + autoAR.InnovationVariance);
Console.Out.WriteLine("Constant = " + autoAR.Constant);
Console.Out.WriteLine();
new PrintMatrix("AR estimates are: ").Print(autoAR.GetAR());

Output

Method of Moments

Order Selected: 3
AIC = 405.981241965022 Variance = 287.269907744347
Constant = 13.7097894010953

AR estimates are:
0
0 1.36757589345393
1 -0.737673445646019
2 0.0782508772709516

Least Squares

Order Selected: 3
AIC = 405.981241965022 Variance = 221.995196605174
Constant = 11.6104946249815

AR estimates are:
0
0 1.55022677857954
1 -1.00394753883711
2 0.205447994348897

Maximum Likelihood

Order Selected: 3
AIC = 405.981241965022 Variance = 218.848375884726
Constant = 11.4178509825012

AR estimates are:
0
0 1.55139136379628
1 -0.998935931377056
2 0.204487453682164

Time Series and Forecasting

ARAutoUnivariate • 857
Example 2 (Forecasting): ARAutoUnivariate

Using the Canadian Lynx data included in TIMSAC-78, ARAutoUnivariate is used to find the minimum AIC autoregressive model using a maximum number of lags of maxlag=20.

This example compares the three different methods for estimating the autoregressive coefficients, and it illustrates the relationship between these estimates and those calculated within the TIMSAC UNIMAR code. As illustrated, the UNIMAR code estimates the coefficients and innovation variance using only the last N-maxlag values in the time series. The other estimation methods use all N-k values, where k is the number of lags with minimum AIC selected by ARAutoUnivariate.

This example also illustrates how to generate forecasts for the observed series values and beyond by setting the backward origin for the forecasts.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;
using PrintMatrix = Imsl.Math.PrintMatrix;
public class ARAutoUnivariateEx2
{
    public static void Main(String[] args)
    {
        /* THE CANDIAN LYNX DATA AS USED IN TIMSAC 1821-1934 */
        double[] y = new double[]{
            0.24300e01, 0.25060e01, 0.27670e01, 0.29400e01, 0.31690e01,
            0.34500e01, 0.35940e01, 0.37740e01, 0.36950e01, 0.34110e01,
            0.27180e01, 0.19910e01, 0.22650e01, 0.24460e01, 0.26120e01,
            0.33590e01, 0.34290e01, 0.35330e01, 0.32610e01, 0.32610e01,
            0.21790e01, 0.16530e01, 0.18320e01, 0.23280e01, 0.27370e01,
            0.30140e01, 0.33280e01, 0.34040e01, 0.29410e01, 0.25730e01,
            0.25730e01, 0.23890e01, 0.27420e01, 0.32100e01, 0.35200e01,
            0.38280e01, 0.36280e01, 0.28370e01, 0.24060e01, 0.26500e01,
            0.25540e01, 0.28940e01, 0.32020e01, 0.32240e01, 0.33520e01,
            0.31540e01, 0.28780e01, 0.24760e01, 0.23030e01, 0.23600e01,
            0.26710e01, 0.28670e01, 0.31100e01, 0.34490e01, 0.36460e01,
            0.34000e01, 0.29500e01, 0.18630e01, 0.15310e01, 0.16900e01,
            0.17710e01, 0.22740e01, 0.25760e01, 0.31100e01, 0.36500e01,
            0.35430e01, 0.27690e01, 0.20210e01, 0.21850e01, 0.25880e01,
            0.28800e01, 0.31150e01, 0.35400e01, 0.38450e01, 0.38000e01,
            0.35790e01, 0.32640e01, 0.25380e01, 0.25820e01, 0.29070e01,
            0.31420e01, 0.34330e01, 0.35800e01, 0.34900e01, 0.34750e01,
            0.35790e01, 0.28290e01, 0.19090e01, 0.19030e01, 0.20330e01,
            0.23600e01, 0.26010e01, 0.30540e01, 0.33860e01, 0.35530e01,
            0.34680e01, 0.31870e01, 0.27230e01, 0.26860e01, 0.28210e01,
            0.30000e01, 0.32010e01, 0.34240e01, 0.35310e01};
        double[] printOutput = null;
        double[] timsacAR, mmAR, mleAR, lsAR;
        double[] forecasts, residuals;
        double timsacConstant, mmConstant, mleConstant, lsConstant;
        double timsacVar, timsacEquivalentVar, mmVar, mleVar, lsVar;
        int maxlag = 20;
        String[] colLabels = new String[]{
            "TIMSAC", "Method of Moments",};
```
"Least Squares",
"Maximum Likelihood";
String[] colLabels2 = new String[]{"Observed", "Forecast", "Residual"};
PrintMatrixFormat pmf = new PrintMatrixFormat();
PrintMatrix pm = new PrintMatrix();
pmf.SetColumnLabels(colLabels);
pmf.NumberFormat = "0.0000";
Console.Out.WriteLine
    ("Automatic Selection of Minimum AIC AR Model");
Console.Out.WriteLine("\n");
ARAutoUnivariate autoAR = new ARAutoUnivariate(maxlag, y);
autoAR.Compute();
int orderSelected = autoAR.Order;
Console.Out.WriteLine("Minimum AIC Selected=" + autoAR.AIC
    + " with an optimum lag of k= " + autoAR.Order);
Console.Out.WriteLine("\n");
timsacAR = autoAR.GetTimsacAR();
timsacConstant = autoAR.TimsacConstant;
timsacVar = autoAR.TimsacVariance;
lSAR = autoAR.GetAR();
lSConstant = autoAR.Constant;
lSVar = autoAR.InnovationVariance;
autoAR.EstimationMethod =
autoAR.Compute();
mmAR = autoAR.GetAR();
mmConstant = autoAR.Constant;
mmVar = autoAR.InnovationVariance;
autoAR.EstimationMethod =
autoAR.Compute();
mleAR = autoAR.GetAR();
mleConstant = autoAR.Constant;
mleVar = autoAR.InnovationVariance;
printOutput = new double[orderSelected + 1][];
for (int i = 0; i < orderSelected + 1; i++)
{
    printOutput[i] = new double[4];
}
printOutput[0][0] = timsacConstant;
for (int i = 0; i < orderSelected; i++)
    printOutput[i + 1][0] = timsacAR[i];
printOutput[0][1] = mmConstant;
for (int i = 0; i < orderSelected; i++)
    printOutput[i + 1][1] = mmAR[i];
printOutput[0][2] = lSConstant;
for (int i = 0; i < orderSelected; i++)
    printOutput[i + 1][2] = lSAR[i];
printOutput[0][3] = mleConstant;
for (int i = 0; i < orderSelected; i++)
    printOutput[i + 1][3] = mleAR[i];
pmSetTitle("Comparison of AR Estimates");

pm.Print(pmf, printOutput);

/* calculation of equivalent innovation variance using TIMSAC coefficients. The TIMSAC innovation variance is calculated using only N-maxlag observations in the series. The following code calculates the innovation variance using N-k observations in the series with the TIMSAC coefficient. This illustrates that the least squares TIMSAC coefficients will not have the least value for the sum of squared residuals, which is calculated using all N-k observations. */

ARMA armaLS = new ARMA(orderSelected, 0, y);
armaLS.SetARMAInfo(autoAR.TimsacConstant, autoAR.GetTimsacAR(),
    new double[0], autoAR.TimsacVariance);
armaLS.BackwardOrigin = y.Length - orderSelected;
forecasts = armaLS.GetForecast(1);
double sumResiduals = 0.0;
for (int i = 0; i < y.Length - orderSelected; i++)
{
    sumResiduals += (y[i + orderSelected] - forecasts[i]) *
        (y[i + orderSelected] - forecasts[i]);
}
timsacEquivalentVar = sumResiduals / (y.Length - orderSelected - 1);

printOutput = new double[1][1];
for (int i2 = 0; i2 < 1; i2++)
{
    printOutput[i2] = new double[4];
}
printOutput[0][0] = timsacEquivalentVar;

/* the method of moments variance */
printOutput[0][1] = mmVar;
/* the least squares variance */
printOutput[0][2] = lsVar;
/* the maximum likelihood estimate of the variance */
printOutput[0][3] = mleVar;

pm.SetTitle("Comparison of Equivalent Innovation Variances");

pm.Print(pmf, printOutput);

/* FORECASTING - An example of forecasting using the maximum likelihood estimates for the minimum AIC AR model. In this example, forecasts are returned for the last 10 values in the series followed by the forecasts for the next 5 values. */

autoAR.BackwardOrigin = 10;
forecasts = autoAR.GetForecast(15);
residuals = autoAR.GetResiduals();
printOutput = new double[15][1];
for (int i3 = 0; i3 < 15; i3++)
{
    printOutput[i3] = new double[3];
}
for (int i = 0; i < 10; i++)
{
    printOutput[i][0] = y[y.Length - 10 + i];
    printOutput[i][1] = forecasts[i];
    printOutput[i][2] = residuals[i];
}
for (int i = 10; i < 15; i++)
{
    printOutput[i][0] = Double.NaN;
    printOutput[i][1] = forecasts[i];
    printOutput[i][2] = Double.NaN;
}

pmf.FirstRowNumber = 105;
pmf.SetColumnLabels(colLabels2);
pm.SetTitle("Maximum Likelihood Forecasts of Last 10 Values");
pm.Print(pmf, printOutput);

Output

Automatic Selection of Minimum AIC AR Model

Minimum AIC Selected=-296.130132635624 with an optimum lag of k= 11

<table>
<thead>
<tr>
<th></th>
<th>TIMSAC</th>
<th>Method of Moments</th>
<th>Least Squares</th>
<th>Maximum Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0427</td>
<td>1.1679</td>
<td>1.1144</td>
<td>1.1186</td>
</tr>
<tr>
<td>1</td>
<td>1.1813</td>
<td>1.1381</td>
<td>1.1481</td>
<td>1.1664</td>
</tr>
<tr>
<td>2</td>
<td>-0.5516</td>
<td>-0.5061</td>
<td>-0.5331</td>
<td>-0.5419</td>
</tr>
<tr>
<td>3</td>
<td>0.2314</td>
<td>0.2098</td>
<td>0.2757</td>
<td>0.2624</td>
</tr>
<tr>
<td>4</td>
<td>-0.1780</td>
<td>-0.2672</td>
<td>-0.3263</td>
<td>-0.3052</td>
</tr>
<tr>
<td>5</td>
<td>0.0199</td>
<td>0.1112</td>
<td>0.1685</td>
<td>0.1519</td>
</tr>
<tr>
<td>6</td>
<td>-0.0626</td>
<td>-0.1246</td>
<td>-0.1643</td>
<td>-0.1460</td>
</tr>
<tr>
<td>7</td>
<td>0.0286</td>
<td>0.0693</td>
<td>0.0728</td>
<td>0.0582</td>
</tr>
<tr>
<td>8</td>
<td>-0.0507</td>
<td>-0.0419</td>
<td>-0.0305</td>
<td>-0.0310</td>
</tr>
<tr>
<td>9</td>
<td>0.1999</td>
<td>0.1366</td>
<td>0.1509</td>
<td>0.1579</td>
</tr>
<tr>
<td>10</td>
<td>0.1618</td>
<td>0.1828</td>
<td>0.1935</td>
<td>0.1995</td>
</tr>
<tr>
<td>11</td>
<td>-0.3391</td>
<td>-0.3101</td>
<td>-0.3414</td>
<td>-0.3375</td>
</tr>
</tbody>
</table>

Comparison of Equivalent Innovation Variances

<table>
<thead>
<tr>
<th></th>
<th>TIMSAC</th>
<th>Method of Moments</th>
<th>Least Squares</th>
<th>Maximum Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0377</td>
<td>0.0427</td>
<td>0.0369</td>
<td>0.0362</td>
</tr>
</tbody>
</table>

Maximum Likelihood Forecasts of Last 10 Values

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>3.5530</td>
<td>3.4387</td>
<td>0.1143</td>
</tr>
<tr>
<td>106</td>
<td>3.4680</td>
<td>3.4801</td>
<td>-0.0121</td>
</tr>
<tr>
<td>107</td>
<td>3.1870</td>
<td>2.9244</td>
<td>0.2626</td>
</tr>
<tr>
<td>108</td>
<td>2.7230</td>
<td>2.7026</td>
<td>0.0204</td>
</tr>
<tr>
<td>109</td>
<td>2.6860</td>
<td>2.5558</td>
<td>0.1302</td>
</tr>
<tr>
<td>110</td>
<td>2.8210</td>
<td>2.7852</td>
<td>0.0358</td>
</tr>
<tr>
<td>111</td>
<td>3.0000</td>
<td>2.9492</td>
<td>0.0508</td>
</tr>
<tr>
<td>112</td>
<td>3.2010</td>
<td>3.1861</td>
<td>0.0149</td>
</tr>
<tr>
<td>113</td>
<td>3.4240</td>
<td>3.3855</td>
<td>0.0385</td>
</tr>
<tr>
<td>114</td>
<td>3.5310</td>
<td>3.5272</td>
<td>0.0038</td>
</tr>
<tr>
<td>115</td>
<td>NaN</td>
<td>3.4465</td>
<td>NaN</td>
</tr>
<tr>
<td>116</td>
<td>NaN</td>
<td>3.1947</td>
<td>NaN</td>
</tr>
<tr>
<td>117</td>
<td>NaN</td>
<td>2.8289</td>
<td>NaN</td>
</tr>
<tr>
<td>118</td>
<td>NaN</td>
<td>2.4918</td>
<td>NaN</td>
</tr>
</tbody>
</table>

Time Series and Forecasting  ARAutoUnivariate • 861
ARAutoUnivariate.ParamEstimation Enumeration

public enumeration Imsl.Stat.ARAutoUnivariate.ParamEstimation

Parameter Estimation procedures.

Fields

**LeastSquares**


Description

Indicates autoregressive parameters are estimated by a least-squares procedure.

**MaximumLikelihood**


Description

Indicates autoregressive parameters are estimated by a maximum likelihood procedure.

**MethodOfMoments**


Description

Indicates autoregressive parameters are estimated by a method of moments procedure.

ARSeasonalFit Class

public class Imsl.Stat.ARSeasonalFit

Estimates the optimum seasonality parameters for a time series using an autoregressive model, AR(\(p\)), to represent the time series.

ARMA time series modeling assumes the time series is stationary. Seasonal trends and cycles violate this assumption, which can lead to inaccurate predictions. However, in many cases the nonstationary series
can be transformed into a stationary series by first differencing the series. For example, if the correlation is strong from one period to the next, the series might be differenced by a lag of 1. Instead of fitting a model to the original series \(Z_t\), the model is fitted to the transformed series:

\[ W_t = Z_t - Z_{t-1} \]

Higher order lags or differences are warranted if the series has cycles every 4 or 13 weeks. Class ARSeasonalFit is designed to help identify the optimum differencing for a series with seasonal trends or cycles.

ARSeasonalFit assumes the original series has no missing values, is equally spaced in time and is not centered before computing the optimum differencing. However, by default the transformed series is centered using the mean of that series. Users can change this default setting with the Center property. If Center is set to None (p. 873) the series is not centered, if set to Mean (p. 873) the series is centered using the mean of the series, and if it is set to Median (p. 873), the series is centered using the median of the series. If Center is set to Mean or Median then the differenced series, \(W_t\) is centered before determination of minimum AIC and optimum lag.

For every combination of rows in SInitial and DInitial, the series \(Z_t\) is converted to the seasonally adjusted series using the following computation

\[
W_t(s,d) = \Delta_{s_1} \Delta_{s_2} \cdots \Delta_{s_m} Z_t = \prod_{i=1}^{m} \sum_{j=0}^{d_i} \left( d_i \right) (-1)^j B^{i:s_i} Z_t
\]

where \(s := (s_1, \ldots, s_m), d := (d_1, \ldots, d_m)\) represent specific rows of arrays SInitial and DInitial respectively, and \(m\) is the number of differences, or \(m = \text{SInitial.GetLength}(1)\).

This transformation of the series \(Z_t\) to \(W_t(s,d)\) is computed using the Difference class. After this transformation the transformed series

\[ W_t(s,d) \]

is centered, unless None is specified, and the ARAutoUnivariate (p. 841) class is used to automatically determine the optimum lag for an AR(\(p\)) representation for \(W_t(s,d)\).

This procedure is repeated for every possible combination of rows in SInitial and DInitial. The series with the minimum AIC is identified as the optimum representation and returned in the methods and properties AROrder (p. 864), GetOptimumS (p. 868), GetOptimumD (p. 867), AIC (p. 863), and GetAR (p. 866). The transformed series with the minium AIC can be retrieved from the GetTransformedTimeSeries (p. 869) method.

### Properties

**AIC**

public double AIC {get; }

**Description**
The final estimate for Akaike’s Information Criterion (AIC) at the optimum.

**Property Value**

A double containing the optimum AIC estimate. \(\text{AIC} \approx -2\ln(L) + 2p\), where \(L\) is the value of the maximum likelihood function evaluated at the parameter estimates.
**AROrder**

`public int AROrder {get; }`

**Description**

The optimum number of lags, $p$, for the optimum autoregressive AR($p$) model. This is the value of $p$ for the transformed series, $W_t$.

**Property Value**

An `int` containing the optimum number of lags in the autoregressive model used to fit the transformed series $W_t$.

**Center**

`public Imsl.Stat.ARSeasonalFit.CenterMethod Center {get; set; }`

**Description**

The setting for centering the input time series.

**Property Value**

A `CenterMethod` containing the setting for Center, either *None* (p. 873), *Mean* (p. 873), or *Median* (p. 873).

**Remarks**

By default, Center=CenterMethod.Mean

**Exclude**

`public bool Exclude {get; set; }`

**Description**

Controls whether to exclude or replace the intial values in the transformed series.

**Property Value**

A `bool` value that controls whether values in $W_t$ that cannot be calculated are dropped or set to missing. Missing values are set to `Double.NaN`.

**Remarks**

If Exclude is `true`, then intial values in the transformed series that cannot be computed are set to missing, NaN. This ensures that the length of the transformed series $W_t$ is equal to the length of the time series, $z$.Length. If Exclude is set to `false`, then intial values in the transformed series $W_t$ that cannot be computed are removed. This makes the length of the transformed series $W_t$ equal to $z$.Length–NLost where NLost is the number of lost values obtained from property NLost (p. 865). By default, Exclude=true.

**Maxlag**

`public int Maxlag {get; }`

**Description**

The maximum lag used to fit the AR($p$) model.
**Property Value**

An int scalar containing the maximum lag allowed when fitting an AR(p) model.

**NLost**

```java
public int NLost {get; }
```

**Description**

The number of values in the initial part of the series lost to differencing.

**Property Value**

An int containing the number of values in the initial part of the series lost to differencing. These lost values will be set to missing or dropped in the transformed series, depending upon the setting for exclude.

**Exceptions**

- `Imsl.Stat.Matrix SingularException` is thrown if the input matrix is singular.
- `Imsl.Stat.TooManyCallsException` is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- `Imsl.Stat.IncreaseErrRelException` is thrown if the bound for the relative error is too small.
- `Imsl.Stat.NewInitialGuessException` is thrown if the iteration has not made good progress.
- `Imsl.Stat.IllConditionedException` is thrown if the problem is ill-conditioned.
- `Imsl.Stat.TooManyIterationsException` is thrown if the maximum number of iterations is exceeded.
- `Imsl.Stat.TooManyFunctionEvaluationsException` is thrown if the maximum number of function evaluations is exceeded.
- `Imsl.Stat.TooManyJacobianEvalException` is thrown if the maximum number of Jacobian evaluations is exceeded.
- `Imsl.Stat.NonStationaryException` is thrown if the final maximum likelihood estimates for the time series are nonstationary.
- `Imsl.Stat.SingularTriangularMatrixException` is thrown if the input triangular matrix is singular.

**Constructor**

```java
public ARSeasonalFit(int maxlag, int[,] SInitial, double[] z)
```

**Description**

Constructor for `ARSeasonalFit`. 
Parameters

maxlag – An int scalar specifying the maximum lag allowed when fitting an AR(\(p\)) model.
SInitial – An int matrix where each row represents seasonal differences to evaluate. The number of columns in SInitial represent the number of differences to perform. All values of SInitial must be greater than zero.
z – A double array containing the time series.

Methods

Compute

public void Compute()

Description

Computes the minimum AIC and optimum values for \(s\) and \(d\) based upon the candidates provided in SInitial and DInitial, and computes the values for the transformed series, \(W_t(s,d)\).

Remarks

Warnings are printed if a row in SInitial is skipped due to too many observations lost in the differenced series or if problems occurred computing the optimum AIC using the differenced series.

Exceptions

Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalsException is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.
Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.

GetAR

public double[] GetAR()
Description
Returns the final autoregressive parameter estimates at the optimum in the transformed series $W_t$.

Returns
A double array containing the estimates for the autoregressive parameters.

Exceptions
- `Imsl.Stat.MatrixSingularException` is thrown if the input matrix is singular.
- `Imsl.Stat.TooManyCallsException` is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- `Imsl.Stat.IncreaseErrRelException` is thrown if the bound for the relative error is too small.
- `Imsl.Stat.NewInitialGuessException` is thrown if the iteration has not made good progress.
- `Imsl.Stat.IllConditionedException` is thrown if the problem is ill-conditioned.
- `Imsl.Stat.TooManyIterationsException` is thrown if the maximum number of iterations is exceeded.
- `Imsl.Stat.TooManyFunctionEvaluationsException` is thrown if the maximum number of function evaluations is exceeded.
- `Imsl.Stat.TooManyJacobianEvalException` is thrown if the maximum number of Jacobian evaluations is exceeded.
- `Imsl.Stat.NonStationaryException` is thrown if the final maximum likelihood estimates for the time series are nonstationary.
- `Imsl.Stat.SingularTriangularMatrixException` is thrown if the input triangular matrix is singular.

---

GetDInitial
public int[,] GetDInitial()

Description
Returns the candidate values for $d$ to evaluate.

Returns
An int matrix containing the candidate values for $d$ to evaluate.

GetOptimumD
public int[] GetOptimumD()

Description
Returns the optimum values for $d$ selected among the candidates in DInitial.

Returns
An int array of length DInitial.GetLength(1) containing the optimum values for $d$ selected among the candidates in dInitial.
Exceptions
Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.
Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.

GetOptimumS
public int[] GetOptimumS()

Description
Returns the optimum values for s selected among the candidates in SInitial.

Returns
An int array of length SInitial.GetLength(1) containing the optimum values for s selected among the candidates in SInitial.

Exceptions
Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.
Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.

GetSInitial
public int[,] GetSInitial()

Description
Returns the candidate values for \( s \) to evaluate.

Returns
An int matrix containing the candidate values for \( s \) to evaluate.

GetTimeSeries
public double[] GetTimeSeries()

Description
Returns the time series.

Returns
A double array containing the time series values.

GetTransformedTimeSeries
public double[] GetTransformedTimeSeries()

Description
Returns the transformed series, \( W_t(s, d) \).

Returns
A double array of length \( z.\text{Length} \) or \( z.\text{Length}-\text{NLost} \), depending upon the setting for the Exclude property.

Remarks
\( W_t(s, d) \) is an array of length \( z.\text{Length} \) or \( z.\text{Length}-\text{NLost} \) containing the optimum seasonally adjusted, autoregressive series, where \( \text{NLost} \) is the first lost observations in this series that are dropped due to differencing. If the missing values are not dropped the first \( \text{NLost} \) values of \( W_t \) will be set to Double.NaN. The \( \text{NLost} \) (p. 865) property can be used to obtain the number of lost observations.

The seasonal adjustment is done by selecting optimum values for \( d_1, \ldots, d_m, s_1, \ldots, s_m \) and \( p \) in the AR model, where \( m \) is number of differences, \( m=\text{SInitial.GetLength(1)} \):

\[
\phi_p(B)(\Delta_{s_1}^{d_1}\Delta_{s_2}^{d_2}\ldots\Delta_{s_m}^{d_m}Z_t - \mu) = a_t,
\]

where \( \{Z_t\} \) is the original time series, \( B \) is the backward shift operator defined by \( B^kZ_t = Z_{t-k} \), with \( k \geq 0 \), \( a_t \) is Gaussian white noise with \( E[a_t] = 0 \) and \( \text{Var}[a_t] = \sigma^2 \), \( \phi_p(B) = 1 - \phi_1B - \phi_2B^2 - \cdots - \phi_pB^p \), \( 0 \leq p \leq \text{maxlag} \), \( \Delta_{s}^{d} = (1-B^d)^s \), with \( s > 0, d \geq 0 \), and \( \mu \) is a centering parameter for the differenced series.

Time Series and Forecasting

ARSeasonalFit • 869
Exceptions

- Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
- Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
- Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
- Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
- Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
- Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
- Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
- Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.
- Imsl.Stat.SingularTriangularMatrixException is thrown if the input triangular matrix is singular.

SetDInitial

```csharp
public void SetDInitial(int[,] DInitial)
```

Description

Sets the candidate values for selecting the optimum seasonal adjustment prior to calling the compute method.

Parameter

- `DInitial` - An `int[,]` array of candidate values for `d` to evaluate. All values must be non-negative. `DInitial` must have the same number of differences (columns) as `SInitial`. By default, `DInitial` is initialized to all ones.

Example: ARSeasonalFit

Consider the Airline Data (Box, Jenkins and Reinsel 1994, p. 547) consisting of the monthly total number of international airline passengers from January 1949 through December 1960. Class ARSeasonalFit is used to compute the optimum seasonality representation of the adjusted series

\[
W_t(s,d) = \Delta_{s,1} \Delta_{d,2} Z_t = (1 - B^s)^d (1 - B^{12})^{d_z} Z_t,
\]

where

- \( s = (1, 1) \)

or

- \( s = (1, 12) \)
and 
\[ d = (1, 1). \]

As differenced series with minimum AIC,
\[ W_t = \Delta_1^1 \Delta_1^2 Z_t = (Z_t - Z_{t-12}) - (Z_{t-1} - Z_{t-13}), \]
is identified.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;
using PrintMatrix = Imsl.Math.PrintMatrix;
using Imsl;

public class ARSeasonalFitEx1
{
    public static void Main(String[] args)
    {
        double[] z =
            new double[] {112, 118, 132, 129, 135, 148, 148, 136, 119,
                          104, 118, 115, 126, 141, 135, 125, 149, 170,
                          170, 158, 133, 114, 140, 145, 150, 178, 163,
                          172, 178, 199, 199, 184, 162, 146, 166, 171,
                          180, 193, 181, 183, 218, 230, 242, 209, 191,
                          172, 194, 196, 196, 236, 235, 229, 243, 264,
                          272, 237, 211, 180, 201, 204, 188, 235, 227,
                          234, 264, 302, 293, 259, 229, 203, 229, 242,
                          233, 267, 269, 270, 315, 364, 347, 312, 274,
                          237, 278, 284, 277, 317, 313, 318, 374, 413,
                          355, 422, 465, 467, 404, 347, 305, 336, 340,
                          318, 362, 348, 363, 385, 491, 505, 404, 359,
                          310, 337, 360, 342, 406, 396, 420, 472, 548,
                          559, 463, 407, 362, 405, 417, 391, 419, 461,
                          472, 535, 622, 606, 508, 461, 390, 432};

        int[,] sInit = {{1, 1}, {1, 12}};
        ARSeasonalFit seasFit = new ARSeasonalFit(10, sInit, z);
        seasFit.Compute();
        Console.Out.WriteLine("NLost = " + seasFit.NLost);
        Console.Out.WriteLine("aic = " + seasFit.AIC);
        new PrintMatrix("Best Periods (Optimum S)").Print(seasFit.GetOptimumS());
        new PrintMatrix("Best Orders (Optimum D)").Print(seasFit.GetOptimumD());
        Console.Out.WriteLine("Best AR order selected = " + seasFit.AROrder);
        new PrintMatrix("AR Coefficients").Print(seasFit.GetAR());

        Console.Out.WriteLine("\n");
        double[] w = seasFit.GetTransformedTimeSeries();
        double[][] pack = new double[30][];
        for (int i = 0; i < 30; i++)
            { pack[i] = new double[2];
```
pack[i][0] = z[i];
pack[i][1] = w[i];
}
PrintMatrix pm = new PrintMatrix();
String str = "First 30 elements of the original time series and " +
"differenced series";
pm.SetTitle(str);
PrintMatrixFormat fmt = new PrintMatrixFormat();
fmt.SetColumnLabels(new String[]{"Original", "Differenced"});
pm.Print(fmt, pack);
}

Output

NLost = 13
aic = 606.397245653491
Best Periods (Optimum S)
 0
 1
 1 12

Best Orders (Optimum D)
 0
 1
 1 1

Best AR order selected = 1
AR Coefficients
 0
 0 -0.309834454941166

First 30 elements of the original time series and differenced series
Original   Differenced
0       112   NaN
1       118   NaN
2       132   NaN
3       129   NaN
4       121   NaN
5       135   NaN
6       148   NaN
7       148   NaN
8       136   NaN
9       119   NaN
10      104   NaN
11      118   NaN
12      115   NaN
13      126   5
14      141   1
15      135   -3
16      125   -2
17      149   10
18      170   8
19      170   0

872 • ARSeasonalFit

IMSL C# Numerical Library
ARSeasonalFit.CenterMethod Enumeration

public enumeration Imsl.Stat.ARSeasonalFit.CenterMethod

Methods for centering the input time series.

Fields

Mean

public Imsl.Stat.ARSeasonalFit.CenterMethod Mean

Description
Indicates the transformed series should be centered using the average of the differenced series.

Median

public Imsl.Stat.ARSeasonalFit.CenterMethod Median

Description
Indicates the transformed series should be centered using the median of the differenced series.

None

public Imsl.Stat.ARSeasonalFit.CenterMethod None

Description
Indicates the transformed series should not be centered.
ARMA Class

public class Imsl.Stat.ARMAMethod

Computes least-square estimates of parameters for an ARMA model.

Class ARMA computes estimates of parameters for a nonseasonal ARMA model given a sample of observations, \( \{W_t\} \), for \( t = 1, 2, \ldots, n \), where \( n = z.\text{Length} \). There are two methods, method of moments and least squares, from which to choose. The default is method of moments.

Two methods of parameter estimation, method of moments and least squares, are provided. The user can choose a method using the Method property. If the user wishes to use the least-squares algorithm, the preliminary estimates are the method of moments estimates by default. Otherwise, the user can input initial estimates by using the SetInitialEstimates method. The following table lists the appropriate methods and properties for both the method of moments and least-squares algorithm:

<table>
<thead>
<tr>
<th>Least Squares</th>
<th>Both Method of Moment and Least Squares</th>
</tr>
</thead>
<tbody>
<tr>
<td>Center</td>
<td>Method</td>
</tr>
<tr>
<td>ARLags</td>
<td>RelativeError</td>
</tr>
<tr>
<td>MALags</td>
<td>MaxIterations</td>
</tr>
<tr>
<td>Backcasting</td>
<td>ConvergenceTolerance</td>
</tr>
<tr>
<td>SetInitialEstimates</td>
<td>Mean</td>
</tr>
<tr>
<td>Residual</td>
<td>AutoCovariance</td>
</tr>
<tr>
<td>SSResidual</td>
<td>Variance</td>
</tr>
<tr>
<td>ParamEstimatesCovariance</td>
<td>Constant</td>
</tr>
<tr>
<td>AR</td>
<td></td>
</tr>
<tr>
<td>MA</td>
<td></td>
</tr>
</tbody>
</table>

Method of Moments Estimation

Suppose the time series \( \{Z_t\} \) is generated by an ARMA \( (p, q) \) model of the form

\[
\phi(B)Z_t = \theta_0 + \theta(B)A_t,
\]

for \( t \in \{0, \pm 1, \pm 2, \ldots\} \)

Let \( \hat{\mu} = \text{Mean} \) be the estimate of the mean \( \mu \) of the time series \( \{Z_t\} \), where \( \hat{\mu} \) equals the following:

\[
\hat{\mu} = \begin{cases} 
\mu & \text{for } \mu \text{ known} \\
\frac{1}{n} \sum_{t=1}^{n} Z_t & \text{for } \mu \text{ unknown}
\end{cases}
\]
The autocovariance function is estimated by

\[ \hat{\sigma}(k) = \frac{1}{n} \sum_{t=1}^{n-k} (Z_t - \hat{\mu})(Z_{t+k} - \hat{\mu}) \]

for \( k = 0, 1, \ldots, K \), where \( K = p + q \). Note that \( \hat{\sigma}(0) \) is an estimate of the sample variance.

Given the sample autocovariances, the function computes the method of moments estimates of the autoregressive parameters using the extended Yule-Walker equations as follows:

\[ \hat{\Sigma} \hat{\phi} = \hat{\sigma} \]

where

\[ \hat{\phi} = (\hat{\phi}_1, \ldots, \hat{\phi}_p)^T \]

\[ \hat{\Sigma}_{ij} = \hat{\sigma}(|q+i-j|), \ i, j = 1, \ldots, p \]

\[ \hat{\sigma}_i = \hat{\sigma}(q+i), \ i = 1, \ldots, p \]

The overall constant \( \theta_0 \) is estimated by the following:

\[ \hat{\theta}_0 = \begin{cases} \hat{\mu} & \text{for } p = 0 \\ \hat{\mu} \left(1 - \sum_{i=1}^{p} \hat{\phi}_i\right) & \text{for } p > 0 \end{cases} \]

The moving average parameters are estimated based on a system of nonlinear equations given \( K = p + q + 1 \) autocovariances, \( \sigma(k) \) for \( k = 1, \ldots, K \), and \( p \) autoregressive parameters \( \phi_i \) for \( i = 1, \ldots, p \).

Let \( Z'_t = \phi(B)Z_t \). The autocovariances of the derived moving average process \( Z'_t = \theta(B)\lambda_t \) are estimated by the following relation:

\[ \hat{\sigma}'(k) = \begin{cases} \hat{\sigma}(k) & \text{for } p = 0 \\ \sum_{i=0}^{p} \sum_{j=0}^{p} \hat{\phi}_i \hat{\phi}_j (\hat{\sigma}(|k+i-j|)) & \text{for } p \geq 1, \hat{\phi}_0 = -1 \end{cases} \]

The iterative procedure for determining the moving average parameters is based on the relation
\[
\sigma(k) = \begin{cases} 
(1 + \theta_1^2 + \ldots + \theta_q^2) \sigma_A^2 & \text{for } k = 0 \\
(-\theta_k + \theta_1 \theta_{k-1} + \ldots + \theta_{q-k} \theta_q) \sigma_A^2 & \text{for } k \geq 1
\end{cases}
\]

where \(\sigma(k)\) denotes the autocovariance function of the original \(Z_t\) process.

Let \(\tau = (\tau_0, \tau_1, \ldots, \tau_q)^T\) and \(f = (f_0, f_1, \ldots, f_q)^T\), where

\[
\tau_j = \begin{cases} 
\sigma_A & \text{for } j = 0 \\
-\theta_j / \tau_0 & \text{for } j = 1, \ldots, q
\end{cases}
\]

and

\[
f_j = \sum_{i=0}^{q-j} \tau_i \tau_{i+j} - \sigma'(j) & \text{for } j = 0, 1, \ldots, q
\]

Then, the value of \(\tau\) at the \((i + 1)\)-th iteration is determined by the following:

\[
\tau^{i+1} = \tau^i - (T^i)^{-1} f^i
\]

The estimation procedure begins with the initial value

\[
\tau^0 = (\sqrt{\hat{\sigma}(0)}, \ 0, \ldots, 0)^T
\]

and terminates at iteration \(i\) when either \(\|f^i\|\) is less than \text{RelativeError} or \(i\) equals \text{MaxIterations}.

The moving average parameter estimates are obtained from the final estimate of \(\tau\) by setting

\[
\hat{\theta}_j = -\tau_j / \tau_0 & \text{for } j = 1, \ldots, q
\]

The random shock variance is estimated by the following:

\[
\hat{\sigma}_A^2 = \begin{cases} 
\hat{\sigma}(0) - \sum_{i=1}^p \hat{\phi}_i \hat{\sigma}(i) & \text{for } q = 0 \\
\tau_0^2 & \text{for } q \geq 0
\end{cases}
\]

See Box and Jenkins (1976, pp. 498-500) for a description of a function that performs similar computations.

**Least-squares Estimation**

Suppose the time series \(\{Z_t\}\) is generated by a nonseasonal ARMA model of the form,
\[ \phi(B)(Z_t - \mu) = \theta(B)A_t \text{ for } t \in \{0, \pm 1, \pm 2, \ldots \} \]

where \( B \) is the backward shift operator, \( \mu \) is the mean of \( Z_t \), and

\[
\phi(B) = 1 - \phi_1 B \phi(1) - \phi_2 B \phi(2) - \ldots - \phi_p B \phi(p) \quad \text{for } p \geq 0
\]

\[
\theta(B) = 1 - \theta_1 B \theta(1) - \theta_2 B \theta(2) - \ldots - \theta_q B \theta(q) \quad \text{for } q \geq 0
\]

with \( p \) autoregressive and \( q \) moving average parameters. Without loss of generality, the following is assumed:

\[
1 \leq l_\phi(1) \leq l_\phi(2) \leq \ldots \leq l_\phi(p)
\]

\[
1 \leq l_\theta(1) \leq l_\theta(2) \leq \ldots \leq l_\theta(q)
\]

so that the nonseasonal ARMA model is of order \((p', q')\), where \( p' = l_\theta(p) \) and \( q' = l_\theta(q) \). Note that the usual hierarchical model assumes the following:

\[
l_\phi(i) = i, 1 \leq i \leq p
\]

\[
l_\theta(j) = j, 1 \leq j \leq q
\]

Consider the sum-of-squares function

\[
S_T(\mu, \phi, \theta) = \sum_{t=1}^{T} [A_t]^2
\]

where

\[
[A_t] = E[A_t | (\mu, \phi, \theta, Z)]
\]
and $T$ is the backward origin. The random shocks $\{A_t\}$ are assumed to be independent and identically distributed

$$N(0, \sigma^2_A)$$

random variables. Hence, the log-likelihood function is given by

$$l(\mu, \phi, \theta, \sigma^2_A) = f(\mu, \phi, \theta) - n \ln(\sigma_A) - \frac{S_T(\mu, \phi, \theta)}{2\sigma^2_A}$$

where $f(\mu, \phi, \theta)$ is a function of $\mu$, $\phi$, and $\theta$.

For $T = 0$, the log-likelihood function is conditional on the past values of both $Z_t$ and $A_t$ required to initialize the model. The method of selecting these initial values usually introduces transient bias into the model (Box and Jenkins 1976, pp. 210-211). For $T = \infty$, this dependency vanishes, and estimation problem concerns maximization of the unconditional log-likelihood function. Box and Jenkins (1976, p. 213) argue that

$$S_\infty(\mu, \phi, \theta) / (2\sigma^2_A)$$

dominates

$$l(\mu, \phi, \theta, \sigma^2_A)$$

The parameter estimates that minimize the sum-of-squares function are called least-squares estimates. For large $n$, the unconditional least-squares estimates are approximately equal to the maximum likelihood-estimates.

In practice, a finite value of $T$ will enable sufficient approximation of the unconditional sum-of-squares function. The values of $[A_T]$ needed to compute the unconditional sum of squares are computed iteratively with initial values of $Z_t$ obtained by back forecasting. The residuals (including backcasts), estimate of random shock variance, and covariance matrix of the final parameter estimates also are computed. ARIMA parameters can be computed by using Difference with ARMA.

**Forecasting**

The Box-Jenkins forecasts and their associated probability limits for a nonseasonal ARMA model are computed given a sample of $n = z$. Length, $\{Z_t\}$ for $t = 1, 2, \ldots, n$.

Suppose the time series $Z_t$ is generated by a nonseasonal ARMA model of the form

$$\phi(B)Z_t = \theta_0 + \theta(B)A_t$$
for \( t \in \{0, \pm 1, \pm 2, \ldots \} \), where \( B \) is the backward shift operator, \( \theta_0 \) is the constant, and

\[
\phi(B) = 1 - \phi_1 B^{l_\phi(1)} - \phi_2 B^{l_\phi(2)} - \ldots - \phi_p B^{l_\phi(p)}
\]

\[
\theta(B) = 1 - \theta_1 B^{l_\theta(1)} - \theta_2 B^{l_\theta(2)} - \ldots - \theta_q B^{l_\theta(q)}
\]

with \( p \) autoregressive and \( q \) moving average parameters. Without loss of generality, the following is assumed:

\[
1 \leq l_\phi(1) \leq l_\phi(2) \leq \ldots \leq l_\phi(p)
\]

\[
1 \leq l_\theta(1) \leq l_\theta(2) \leq \ldots \leq l_\theta(q)
\]

so that the nonseasonal ARMA model is of order \((p', q')\), where \( p' = l_\phi(p) \) and \( q' = l_\theta(q) \). Note that the usual hierarchical model assumes the following:

\[
l_\phi(i) = i, 1 \leq i \leq p
\]

\[
l_\theta(j) = j, 1 \leq j \leq q
\]

The Box-Jenkins forecast at origin \( t \) for lead time \( l \) of \( Z_{t+1} \) is defined in terms of the difference equation

\[
\hat{Z}_t(l) = \theta_0 + \phi_1 \left[ Z_{t+l - l_\phi(1)} \right] + \ldots + \phi_p \left[ Z_{t+l - l_\phi(p)} \right]
\]

\[
\quad + \left[ A_{t+l} - \theta_1 A_{t+l - l_\theta(1)} - \ldots - \theta_q A_{t+l - l_\theta(q)} \right]
\]

where the following is true:

\[
[Z_{t+k}] = \begin{cases} 
Z_{t+k} & \text{for } k = 0, -1, -2, \ldots \ 
\hat{Z}_t(k) & \text{for } k = 1, 2, \ldots 
\end{cases}
\]
\[ A_{t+k} = \begin{cases} Z_{t+k} \hat{Z}_{t+k-1}(1) & \text{for } k = 0, -1, -2, \ldots \\ 0 & \text{for } k = 1, 2, \ldots \end{cases} \]

The 100(1 – \( \alpha \)) percent probability limits for \( Z_{t+l} \) are given by

\[ Z_t(l) \pm z_{\alpha/2} \left\{ 1 + \sum_{j=1}^{l-1} \psi_j^2 \right\}^{1/2} \sigma_A \]

where \( z_{\alpha/2} \) is the 100(1 – \( \alpha/2 \)) percentile of the standard normal distribution

\[ \sigma_A^2 \]

and

\[ \{ \psi_j^2 \} \]

are the parameters of the random shock form of the difference equation. Note that the forecasts are computed for lead times \( l = 1, 2, \ldots, L \) at origins \( t = (n - b), (n - b + 1), \ldots, n \), where \( L = \text{nForecast} \) and \( b = \text{BackwardOrigin} \).

The Box-Jenkins forecasts minimize the mean-square error

\[ E \left[ Z_{t+l} - \hat{Z}_t(l) \right]^2 \]

Also, the forecasts can be easily updated according to the following equation:

\[ \hat{Z}_{t+1}(l) = \hat{Z}_t(l+1) + \psi_l A_{t+1} \]

This approach and others are discussed in Chapter 5 of Box and Jenkins (1976).

**Properties**

**BackwardOrigin**

```csharp
public int BackwardOrigin { get; set; }
```

**Description**

The maximum backward origin.
Property Value
An int specifying the maximum backward origin.

Remarks
BackwardOrigin must be greater than or equal to 0 and less than or equal to \( z.\text{Length} - \max(\maxar, \maxma) \), where \( \maxar = \max(ARLags[i]) \), \( \maxma = \max(MALags[j]) \), and forecasts at origins \( z.\text{Length} - \text{BackwardOrigin} \) through \( z.\text{Length} \) are generated. By default, BackwardOrigin = 0.

Center
public bool Center {get; set; }

Description
The center option.

Property Value
A bool value specifying whether the time series center occurs about its mean.

Remarks
If Center is set to false, the time series is not centered about its mean. If Center is set to true, the time series is centered about its mean. By Default, Center = false.

Confidence
public double Confidence {get; set; }

Description
The confidence level for calculating confidence limit deviations returned from GetDeviations.

Property Value
A double scalar specifying the confidence level used in computing forecasts confidence intervals.

Remarks
Typical choices for Confidence are 0.90, 0.95, and 0.99. Confidence must be greater than 0.0 and less than 1.0. By default, Confidence = 0.95.

Constant
public double Constant {get; }

Description
The constant parameter estimate.

Property Value
A double scalar containing the constant parameter estimate.

Remarks
Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is NaN.

ConvergenceTolerance
public double ConvergenceTolerance {get; set; }

Time Series and Forecasting
Description
The tolerance level used to determine convergence of the nonlinear least-squares algorithm.

Property Value
A double scalar containing the tolerance level used to determine convergence of the nonlinear least-squares algorithm.

Remarks
ConvergenceTolerance represents the minimum relative decrease in sum of squares between two iterations required to determine convergence. Hence, ConvergenceTolerance must be greater than or equal to 0. By default ConvergenceTolerance = 10^{-20}.

InnovationVariance
public double InnovationVariance {get; }

Description
The variance of the random shock.

Property Value
A double scalar equal to the variance of the random shock.

MaxIterations
public int MaxIterations {get; set; }

Description
The maximum number of iterations.

Property Value
An int scalar specifying the maximum number of iterations allowed in the nonlinear equation solver used in both the method of moments and least-squares algorithms.

Remarks
By default, MaxIterations = 200.

Mean
public double Mean {get; set; }

Description
An update of the mean of the time series z.

Property Value
A double scalar containing the mean of the time series z.

Remarks
If the time series is not centered about its mean, and the least-squares algorithm is used, the mean is not used in parameter estimation. Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is 0.

Method
public Imsl.Stat.ARMASer.ParamEstimation Method {get; set; }

882 • ARMA IMLS C# Numerical Library
Description
The method used to estimate the autoregressive and moving average parameters estimates.

Property Value
A ParamEstimation specifying the method used to estimate the autoregressive and moving average parameters estimates.

Remarks
If ARMA_ParamEstimation_MethodOfMoments is specified, the autoregressive and moving average parameters are estimated by a method of moments procedure.
If ARMA_ParamEstimation_LeastSquares is specified, the autoregressive and moving average parameters are estimated by a least-squares procedure. By default, Method = ARMA_ParamEstimation_MethodOfMoments.

NumberOfProcessors
public int NumberOfProcessors {get; set; }

Description
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

Property Value
An int indicating the maximum possible number of processors to use.

Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

RelativeError
public double RelativeError {get; set; }

Description
The stopping criterion for use in the nonlinear equation solver.

Property Value
A double scalar containing the stopping criterion for use in the nonlinear equation solver used in both the method of moments and least-squares algorithms.

Remarks
By default, RelativeError = 100 * 2.2204460492503131e-16.

SSResidual
public double SSResidual {get; }

Description
The sum of squares of the random shock.
Property Value
A double scalar containing the sum of squares of the random shock, residual[0]^2 + ... + residual[na - 1]^2, where residual is the array returned from the GetResidual method and na = residual.Length.

Remarks
The Compute method must be invoked first before invoking this method. Otherwise, the return value is 0. This property is only applicable using least-squares algorithm.

Variance
public double Variance {get; }

Description
The variance of the time series z.

Property Value
A double scalar containing the variance of the time series z.

Remarks
Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is NaN.

Constructor

ARMA
public ARMA(int p, int q, double[] z)

Description
Constructor for ARMA.

Parameters
p – An int scalar containing the number of autoregressive (AR) parameters.
q – An int scalar containing the number of moving average (MA) parameters.
z – A double array containing the observations.

Exception
System.ArgumentException is thrown if p, q, and z.Length are not consistent.

Methods

Compute
public void Compute()
Description
Computes least-square estimates of parameters for an ARMA model.

Exceptions
Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.

Forecast
public double[,] Forecast(int nForecast)

Description
Computes forecasts and their associated probability limits for an ARMA model.

Parameter
nForecast – An int scalar containing the maximum lead time for forecasts. nForecast must be greater than 0.

Returns
A double matrix of dimensions of nForecast by BackwardOrigin+1 containing the forecasts. Return null if the least-square estimates of parameters is not computed.

GetAR
public double[] GetAR()

Description
Returns the final autoregressive parameter estimates.

Returns
A double array of length p containing the final autoregressive parameter estimates.

Remarks
Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

GetAutoCovariance
public double[] GetAutoCovariance()
Description
Returns the autocovariances of the time series z.

Returns
A double array containing the autocovariances of lag k, where \( k = 1, \ldots, p + q + 1 \).

Remarks
Note that the Compute method must be invoked before this method. Otherwise, the method throws a NullReferenceException exception.

GetDeviations
public double[] GetDeviations()

Description
Returns the deviations for each forecast used for calculating the forecast confidence limits.

Returns
A double array of length nForecast containing the deviations for calculating forecast confidence intervals. The confidence level is specified in the Confidence property.

Remarks
Note that the Forecast method must be invoked first before invoking this method. Otherwise, the method returns null.

GetForecast
public double[] GetForecast(int nForecast)

Description
Returns forecasts

Parameter
nForecast – An input int representing the number of requested forecasts beyond the last value in the series.

Returns
A double array containing the nForecast+BackwardOrigin forecasts. The first BackwardOrigin forecasts are one-step ahead forecasts for the last BackwardOrigin values in the series. The next nForecast values in the returned series are forecasts for the next values beyond the series.

GetMA
public double[] GetMA()

Description
Returns the final moving average parameter estimates.

Returns
A double array of length q containing the final moving average parameter estimates.
Remarks
Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

GetNumberOfBackcasts
public int GetNumberOfBackcasts()

Description
Returns the number of backcasts used to calculate the AR coefficients for the time series z.

Returns
An int containing the number of backcasts calculated.

Remarks
Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is 0.

GetParamEstimatesCovariance
public double[,] GetParamEstimatesCovariance()

Description
Returns the covariances of parameter estimates.

Returns
A double matrix of np by np dimensions, where np = p + q + 1 if z is centered about Mean, and np = p + q if z is not centered, containing the covariances of parameter estimates.

Remarks
The ordering of variables is mean, AR, and MA.

Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

GetPsiWeights
public double[] GetPsiWeights()

Description
Returns the psi weights of the infinite order moving average form of the model.

Returns
A double array of length nForecast containing the psi weights of the infinite order moving average form of the model.

Remarks
Note that the forecast method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

GetResidual
public double[] GetResidual()
Description
Returns the residuals at the final parameter estimate.

Returns
A double array of length $z.Length - \max(\text{arLags}[i]) + \text{length}$ containing the residuals (including backcasts) at the final parameter estimate point in the first $z.Length - \max(\text{arLags}[i]) + \text{nb}$, where $\text{nb}$ is the number of values backcast.

Remarks
Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception. This method is only applicable using least-squares algorithm.

SetARLags
public void SetARLags(int[] arLags)

Description
The order of the autoregressive parameters.

Parameter
arLags – An int array of length $p$ containing the order of the autoregressive parameters.

Remarks
The elements of arLags must be greater than or equal to 1. By default, $\text{arLags} = [1, 2, \ldots, p]$.

SetARMAInfo
public void SetARMAInfo(double constant, double[] ar, double[] ma, double var)

Description
Sets the ARMAInfo object to previously determined values

Parameters
constant – A double scalar equal to the constant term in the ARMA model.
ar – A double array of length $p$ containing estimates of the autoregressive parameters.
ma – A double array of length $q$ containing estimates of the moving average parameters.
var – A double scalar equal to the innovation variance

SetBackcasting
public void SetBackcasting(int maxBackcast, double tolerance)

Description
Sets backcasting option.
Parameters

- **maxBackcast** – An int scalar containing the maximum length of backcasting and must be greater than or equal to 0. By default, maxBackcast = 10.

- **tolerance** – A double scalar containing the tolerance level used to determine convergence of the backcast algorithm. Typically, tolerance is set to a fraction of an estimate of the standard deviation of the time series. By default, tolerance = 0.01 * standard deviation of z.

---

**SetInitialAREstimates**

*public void SetInitialAREstimates(double[] ar)*

**Description**

Sets preliminary autoregressive estimates.

**Parameter**

- **ar** – A double array of length p containing preliminary estimates of the autoregressive parameters.

---

**SetInitialEstimates**

*public void SetInitialEstimates(double[] ar, double[] ma)*

**Description**

Sets preliminary estimates.

**Parameters**

- **ar** – A double array of length p containing preliminary estimates of the autoregressive parameters.
- **ma** – A double array of length q containing preliminary estimates of the moving average parameters.

---

**SetInitialMAEstimates**

*public void SetInitialMAEstimates(double[] ma)*

**Description**

Sets preliminary moving average estimates.

**Parameter**

- **ma** – A double array of length q containing preliminary estimates of the moving average parameters.
Remarks
ma are computed internally if this method is not used. This method is only applicable using least-squares algorithm.

SetMALags
public void SetMALags(int[] maLags)

Description
Sets the order of the moving average parameters.

Parameter
maLags – An int array of length q containing the order of the moving average parameters.

Remarks
The maLags elements must be greater than or equal to 1. By default, maLags = [1, 2, ..., q].

Example 1: ARMA
Consider the Wolfer Sunspot Data (Anderson 1971, p. 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869. The method of moments estimates \( \hat{\theta}_0, \hat{\phi}_1, \hat{\phi}_2 \), and \( \hat{\theta}_1 \) for the ARMA(2, 1) model

\[
z_t = \theta_0 + \phi_1 z_{t-1} + \phi_2 z_{t-2} - \theta_1 A_{t-1} + A_t
\]

where the errors \( A_t \) are independently normally distributed with mean zero and variance

\[
\sigma_A^2
\]

using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class ARMAEx1
{
    public static void Main(String[] args)
    {
        double[] z = new double[] { 100.8, 81.6, 66.5, 34.8, 30.6, 7, 19.8, 92.5, 154.4, 125.9, 84.8, 68.1, 38.5, 22.8, 10.2, 24.1, 82.9, 132, 130.9, 118.1, 89.9, 66.6, 60, 46.9, 41, 21.3, 16, 6.4, 4.1, 6.8, 14.5, 34, 45, 43.1, 47.5, 42.2, 28.1, 10.1, 8.1, 2.5, ...
    
}
0, 1.4, 5, 12.2, 13.9,
35.4, 45.8, 41.1, 30.4, 23.9,
15.7, 6.6, 4, 1.8, 8.5,
16.6, 36.3, 49.7, 62.5, 67,
71, 47.8, 27.5, 8.5, 13.2,
56.9, 121.5, 138.3, 103.2, 85.8,
63.2, 36.8, 24.2, 10.7, 15,
40.1, 61.5, 98.5, 124.3, 95.9,
66.5, 64.5, 54.2, 39, 20.6,
6.7, 4.3, 22.8, 54.8, 93.8,
95.7, 77.2, 59.1, 44, 47,
30.5, 16.3, 7.3, 37.3, 73.9};

ARMA arma = new ARMA(2, 1, z);
arma.Compute();

new PrintMatrix("AR estimates are: ").Print(arma.GetAR());
Console.Out.WriteLine();
new PrintMatrix("MA estimate is: ").Print(arma.GetMA());
}

Output

AR estimates are:
0
0 1.24425777984372
1 -0.575149766040151

MA estimate is:
0
0 -0.124089747872598

Example 2: ARMA

The data for this example are the same as that for Example 1. Preliminary method of moments estimates
are computed by default, and the method of least squares is used to find the final estimates. Note that at
the end of the output, a warning message appears. In most cases, this warning message can be ignored.
There are three general reasons this warning can occur:

1. Convergence is declared using the criterion based on tolerance, but the gradient of the residual
   sum-of-squares function is nonzero. This occurs in this example. Either the message can be
   ignored or ConvergenceTolerance can be reduced to allow more iterations and a slightly more
   accurate solution.

2. Convergence is declared based on the fact that a very small step was taken, but the gradient of the
   residual sum-of-squares function was nonzero. This message can usually be ignored. Sometimes,
   however, the algorithm is making very slow progress and is not near a minimum.

3. Convergence is not declared after 100 iterations.
Trying a smaller value for `ConvergenceTolerance` can help determine what caused the error message.

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class ARMAEx2
{
    public static void Main(String[] args)
    {
        double[] arInit = new double[]{1.24426e0, - 5.75149e-1};
        double[] maInit = new double[]{- 1.24094e-1};
        double[] z = new double[]{100.8, 81.6, 66.5, 34.8, 30.6, 7, 19.8, 92.5, 154.4, 125.9, 84.8, 68.1, 38.5, 22.8, 10.2, 24.1, 82.9, 132, 130.9, 118.1, 89.9, 66.6, 60, 46.9, 41, 21.3, 16, 6.4, 4.1, 6.8, 14.5, 34, 45, 43.1, 47.5, 42.2, 28.1, 10.1, 8.1, 2.5, 0, 1.4, 5, 12.2, 13.9, 35.4, 45.8, 41.1, 30.4, 23.9, 15.7, 6.6, 4, 1.8, 8.5, 16.6, 36.3, 49.7, 62.5, 67, 71, 47.8, 27.5, 8.5, 13.2, 56.9, 121.5, 138.3, 103.2, 85.8, 63.2, 36.8, 24.2, 10.7, 15, 40.1, 61.5, 98.5, 124.3, 95.9, 66.5, 64.5, 54.2, 39, 20.6, 6.7, 4.3, 22.8, 64.8, 93.8, 95.7, 77.2, 59.1, 44, 47, 30.5, 16.3, 7.3, 37.3, 73.9};
        ARMA arma = new ARMA(2, 1, z);
        arma.SetInitialEstimates(arInit, maInit);
        arma.ConvergenceTolerance = 0.125;
        arma.Mean = 46.976;
        arma.Compute();

        new PrintMatrix("AR estimates are: ").Print(arma.GetAR());
        Console.Out.WriteLine();
        new PrintMatrix("MA estimate is: ").Print(arma.GetMA());
    }
}
```

**Output**

```
AR estimates are:
0
0 1.39325700313638
1 -0.73660553488482
```
MA estimate is:

0
0  -0.137145395974998

Imsl.Stat.ARMA: Relative function convergence - Both the scaled actual and predicted reductions in the function are less than or equal to the relative function convergence tolerance "ConvergenceTolerance" = 0.0645856533065147.
Imsl.Stat.ARMA: Least squares estimation of the parameters has failed to converge. Increase "length" and/or "tolerance" and/or "convergence_tolerance". The estimates of the parameters at the last iteration may be used as new starting values.

Example 3: Forecasting

Consider the Wolfer Sunspot Data (Anderson 1971, p. 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869. An ARMA(2,1) model is fitted to these data using the Method of Moments. With BackwardOrigin = 3, the Forecast method is used to obtain forecasts starting from 1866, 1867, 1868, and 1869, respectively. Note that the values in the first row of the matrix returned by this method are the one-step ahead forecasts for 1867, 1868, ..., 1870. The values in the second row are the two-step ahead forecasts for 1868, 1869, ..., 1871, etc.

Method GetForecast is used to compute the one-step ahead forecasts setting BackwardOrigin = 10. This obtains the one-step ahead forecasts for the last 10 observations in the series, i.e. years 1860-1869, plus the next 5 years. The upper 90% confidence limits are computed for these forecasts using the GetDeviations method.

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;

class ARMAEx3
{
    public static void Main(String[] args)
    {
        double[] z = new double[]{ 100.8, 81.6, 66.5, 34.8, 30.6,
                              7, 19.8, 92.5, 154.4, 125.9,
                              84.8, 68.1, 38.5, 22.8, 10.2,
                              24.1, 82.9, 132, 130.9, 118.1,
                              89.9, 66.6, 60, 46.9, 41,
                              21.3, 16, 6.4, 4.1, 6.8,
                              14.5, 34, 45, 43.1, 47.5,
                              42.2, 28.1, 10.1, 8.1, 2.5,
                              0, 1.4, 5, 12.2, 13.9,
                              35.4, 45.8, 41.1, 30.4, 23.9,
                              15.7, 6.6, 4, 1.8, 8.5,
                              16.6, 36.3, 49.7, 62.5, 67,
                              71, 47.8, 27.5, 8.5, 13.2,
                              56.9, 121.5, 138.3, 103.2, 85.8,
                              63.2, 36.8, 24.2, 10.7, 15,
                              40.1, 61.5, 98.5, 124.3, 95.9,
                              66.5, 64.5, 54.2, 39, 20.6,

```
6.7, 4.3, 22.8, 54.8, 93.8,
95.7, 77.2, 59.1, 44, 47,
30.5, 16.3, 7.3, 37.3, 73.9;

```csharp
double[,] printEstimates = new double[1,4];
PrintMatrixFormat pmf = new PrintMatrixFormat();
PrintMatrix pm = new PrintMatrix();
pm.SetColumnSpacing(3);

ARMA arma = new ARMA(2, 1, z);
arma.Compute();

printEstimates[0,0] = arma.Constant;
double[] ar = arma.GetAR();
printEstimates[0,1] = ar[0];
printEstimates[0,2] = ar[1];
double[] ma = arma.GetMA();
printEstimates[0,3] = ma[0];
String[] estimateLabels = {"Constant", "AR(1)", "AR(2)", "MA(1)"};

pmf.SetColumnLabels(estimateLabels);
pmf.NumberFormat = "0.0000";
pm.SetTitle("ARMA ESTIMATES");
pm.Print(pmf, printEstimates);

String[] labels = new String[]{"From 1866",
"From 1867",
"From 1868",
"From 1869"};

pmf.SetColumnLabels(labels);
pmf.FirstRowNumber = 1;
pmf.NumberFormat = "00.0";
arma.BackwardOrigin = 3;
new PrintMatrix("FORECASTS").Print(pmf, arma.Forecast(5));

double[,] printTable = new double[15,4];
/* FORECASTING - An example of forecasting using the ARMA estimates
* In this case, forecasts are returned for the last 10 values in the
* series followed by the forecasts for the next 5 values.
*/
String[] forecastLabels={"Observed", "Forecast",
"Residual", "UCL(90\%)"};

pmf.SetColumnLabels(forecastLabels);
int backOrigin = 10;
int n_forecast = 5;
arma.BackwardOrigin = backOrigin;
arma.Confidence = 0.9;
double[] forecasts = arma.GetForecast(n_forecast);
double[] deviations = arma.GetDeviations();
for(int i=0; i<backOrigin; i++)
{
    printTable[i,0] = z[z.Length-backOrigin+i];
    printTable[i,1] = forecasts[i];
    printTable[i,2] = z[z.Length-backOrigin+i]-forecasts[i];
    printTable[i,3] = forecasts[i] + deviations[0];
}
for(int i=backOrigin; i<n_forecast+backOrigin; i++)
{
```

894 • ARMA

IMSL C# Numerical Library
Output

<table>
<thead>
<tr>
<th>ARMA ESTIMATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FORECASTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>From 1866</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARMA ONE-STEP AHEAD FORECASTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>1860</td>
</tr>
<tr>
<td>1861</td>
</tr>
<tr>
<td>1862</td>
</tr>
<tr>
<td>1863</td>
</tr>
<tr>
<td>1864</td>
</tr>
<tr>
<td>1865</td>
</tr>
<tr>
<td>1866</td>
</tr>
<tr>
<td>1867</td>
</tr>
<tr>
<td>1868</td>
</tr>
<tr>
<td>1869</td>
</tr>
<tr>
<td>1870</td>
</tr>
<tr>
<td>1871</td>
</tr>
<tr>
<td>1872</td>
</tr>
<tr>
<td>1873</td>
</tr>
<tr>
<td>1874</td>
</tr>
</tbody>
</table>

ARMA.ParamEstimation Enumeration

public enumeration Imsl.Stat.ARMA.ParamEstimation

Time Series and Forecasting  ARMA.ParamEstimation • 895
Parameter Estimation procedures.

Fields

**LeastSquares**

public Imsl.Stat.ARMAD. ParamEstimation LeastSquares

**Description**

Indicates autoregressive and moving average parameters are estimated by a least-squares procedure.

**MethodOfMoments**

public Imsl.Stat.ARMAD. ParamEstimation MethodOfMoments

**Description**

Indicates autoregressive and moving average parameters are estimated by a method of moments procedure.

**ARMAEstimateMissing Class**

public class Imsl.Stat.ARMAD. EstimateMissing

Estimates missing values in a time series collected with equal spacing. Missing values can be replaced by these estimates prior to fitting a time series using the ARMA class.

Traditional time series analysis as described by Box, Jenkins and Reinsel (1994) requires the observations be made at equidistant time points \( t_0, t_1, \ldots, t_n \) where \( t_i = t_0 + i \). When observations are missing, ARMA requires that they be replaced with suitable estimates. Class ARMAEstimateMissing offers 4 methods for estimating missing values: Median, CubicSpline, AR.1, and AR.p

The centering method Median estimates the missing observations in a gap by the median of the last four time series values before and the first four values after the gap. If not enough values are available before or after the gap then the number is reduced accordingly. This method is very fast and simple, but its use is limited to stationary ergodic series without outliers and level shifts.

Centering method CubicSpline uses a cubic spline interpolation method to estimate missing values. Here the interpolation is again done over the last four time series values before and the first four values after the gap. The missing values are estimated by the resulting interpolant. This method gives smooth transitions across missing values.

Method AR.1 assumes that the time series before the gap can be approximated using an AR(1) process. If the last observation prior to the gap is made at time point \( t_m \) then this method uses values at \( t_0, t_1, \ldots, t_m \) to compute the one-step-ahead forecast at origin \( t_m \). This value is used to estimate the missing value at time point \( t_m + 1 \). If the value at \( t_m + 2 \) is also missing then the values at time points \( t_0, t_1, \ldots, t_m + 1 \) are

896 • ARMAEstimateMissing IMSL C# Numerical Library
used to recompute the AR(1) model, and then estimate the value at $t_m + 2$ and so on. The coefficient $\phi_1$ in the AR(1) model is computed internally by the method of least squares from class ARMA.

Finally, method AR_p uses an AR(p) model to estimate missing values using a one-step-ahead forecast similar to method AR_1. First, class ARAutoUnivariate, is applied to the time series values just prior to the missing values to determine the optimum $p$ from the set \{0, 1, ..., maxlag\} of possible values and to compute the parameters $\phi_1, ..., \phi_p$ of the resulting AR(p) model. The parameters are estimated by the least squares method based on Householder transformations as described in Kitagawa and Akaike (1978). Denoting the mean of the series $y_{t_0}, y_{t_1}, ..., y_{t_m}$ by $\mu$ the one-step-ahead forecast at origin $t_m$, $\hat{y}_{t_m}(1)$, can be computed by the formula

$$\hat{y}_{t_m}(1) = \mu (1 - \sum_{j=1}^{p} \phi_j) + \sum_{j=1}^{p} \phi_j y_{t_m+1-j}.$$ 

This value is used as an estimate for the missing value at $t_{m+1}$. The procedure starting with ARAutoUnivariate is then repeated for every further missing value in the gap. All four estimation methods treat gaps of missing values in increasing time order.

### Properties

**ConvergenceTolerance**  
public double ConvergenceTolerance {get; set; }  
**Description**  
The convergence tolerance used by the AR_1 and AR_p missing value estimation methods.  
**Property Value**  
A double scalar value. By default, ConvergenceTolerance = 1.0e-09.

**EstimationMethod**  
public Imsl.Stat.ARAutoUnivariate.ParamEstimation EstimationMethod {get; set; }  
**Description**  
The method used for estimating the final autoregressive coefficients for missing value estimation methods AR_1 and AR_p.  
**Property Value**  
An ARAutoUnivariate.ParamEstimation scalar specifying the method used to estimate the autoregressive coefficients. Valid methods are MethodOfMoments, LeastSquares, and MaxLikelihood.  
**Remarks**  

**MaxIterations**  
public int MaxIterations {get; set; }
**Description**
The maximum number of estimation iterations for missing value estimation methods AR\(_1\) and AR\(_p\).

**Property Value**
An int specifying the maximum number of iterations for the maximum likelihood estimation.

**Remarks**
If this limit is exceeded ARMAEstimateMissing stops execution during the Compute method and issues an Imsl.Stat.IterationLimitExceededException. By default, MaxIterations=200.

---

**Maxlag**

```csharp
public int Maxlag {get; set; }
```

**Description**
The maximum number of autoregressive lags when method AR\(_p\) is selected as the missing value estimation method.

**Property Value**
An int scalar value equal to the maximum number of autoregressive lags. Maxlag must be greater than z.Length-5. By default Maxlag=10.

**Remarks**
Maxlag must be greater than z.Length-5. By default Maxlag=10.

---

**Mean**

```csharp
public double Mean {get; set; }
```

**Description**
The mean value used to center the series.

**Property Value**
A double scalar value used to center the series.

**Remarks**
By default the median of the series is used for centering.

---

**MissingValueMethod**

```csharp
public Imsl.Stat.ARMAEstimateMissing.MissingValueEstimation MissingValueMethod {get; set; }
```

**Description**
The current missing value estimation method.

**Property Value**
A MissingValueEstimation representing the estimation method used for estimating the missing values in the time series. Available methods are Median, CubicSpline, AR\(_1\) or AR\(_p\).

**Remarks**
By default, MissingValueMethod=ARMAEstimateMissing.MissingValueEstimation.AR\(_p\).

---

**NumberMissing**

```csharp
public int NumberMissing {get; }
```
Description
The number of missing values in the original series

Property Value
An int scalar containing the number of missing values in the time series.

Relative Error

class public double RelativeError {get; set; }

Description
The relative error used for the MethodOfMoments and LeastSquares estimation methods.

Property Value
A double scalar containing the stopping criterion for use in the nonlinear equation solver used in both the method of moments and least-squares algorithm.

Remarks
By default, RelativeError = 100 * 2.2204460492503131e-16.

Constructor

ARMAEstimateMissing

class public ARMAEstimateMissing(int[] tpoints, double[] z)

Description
Constructor for ARMAEstimateMissing.

Parameters
tpoints – An int array containing the times at which the series values were observed. The values must be strictly increasing. Times for missing values are identified as non-incremental gaps in this series. A gap of missing values in z is assumed when the difference between two consecutive values is greater than one, i.e. T_{i+1} - T_i > 1. The difference is the number of missing values in the gap. The series can have multiple gaps with missing values, but any one gap can have no more than three missing values.
z – A double array containing the values for the time series observed at the times given in the vector tpoints.

Methods

GetCompleteTimes

class public int[] GetCompleteTimes()
Description
Returns an int array of all time points, including values for times with missing values in z.

Returns
An int array of all times from tpoints[0]=1 to tpoints.Length+NumberMissing, where NumberMissing is the number of values removed from the original time series.

GetCompleteTimeSeries
public double[] GetCompleteTimeSeries()

Description
Returns a double precision vector of length tpoints[tpoints.Length-1]-tpoints[0]+1 containing the observed values in the time series z plus estimates for missing values in gaps identified in tpoints.

Returns
A double array of length tpoints[tpoints.Length-1]-tpoints[0]+1 containing the observed values in the time series z plus estimates for missing values in gaps identified in tpoints.

Exceptions
Imsl.Stat.SingularTriangularMatrixException is thrown if the input matrix to ARAutoUnivariate is singular. This can only occur with estimation method AR.p.
Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
Imsl.Stat.NonStationaryException is thrown if the final maximum likelihood estimates for the time series are nonstationary.

GetMissingTimes
public int[] GetMissingTimes()

Description
Returns the times at which missing values were estimated.
Returns
An int array containing the times at which missing values were estimated. If there are no missing values a null array is returned.

Example: ARMAEstimateMissing

The data in this example was artificially generated using an autoregressive time series with a lag of 1, i.e., AR(1). The constant term in the model was set to zero and -0.7 was used for the autoregressive coefficient. The data were generated from a random gaussian distribution with a mean of zero and an innovation variance of 0.51. This series is stationary with Var(Y) = 1.0.

Two hundred values were generated. For this example, six values at times t=130, t=140, t=141, t=160, t=175, and t=176 are removed and designated as missing. ARMAEstimateMissing is used to estimate these missing values using each of its estimation methods. The missing value estimates are compared to the actual values generated in the full series.

As expected, the AR(1) method produced the best missing value estimates in this example, closely followed by the AR(p) method.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;
using PrintMatrix = Imsl.Math.PrintMatrix;
using Imsl;
public class ARMAEstimateMissingEx1
{
    public static void Main(String[] args)
    {
        int i, k;
        int maxlag = 20;
        int n_obs = 194, n_miss = 6;
        int[] missing_index = null;
        int[] tpointsMiss = null;
        int[] tpointsComplete = null;
        double missVar = 0;
        double sum = 0;
        double variance = 0;
        double[] y = null;
        double[] yMiss = null;
        double[] yComplete =
        {
            1.30540, -1.37166, 1.47905, -0.91059, 1.36191,
            -2.16966, 3.11254, -1.99536, 2.29740, -1.82474,
            -0.25445, 0.33519, -0.25480, -0.50574, -0.21429,
            -0.45932, -0.63813, 0.25646, -0.46243, -0.44104,
            0.42733, 0.61102, -0.82417, 1.48537, -1.57733,
            -0.09846, 0.46311, 0.49156, -1.66090, 2.02808,
            -1.45768, 1.36115, -0.65973, 1.13332, -0.86285,
            1.23848, -0.57301, -0.28210, 0.20195, 0.06981,
            0.28454, 0.19745, -0.16490, -1.05019, 0.78652,
            -0.40447, 0.71514, -0.90003, 1.83604, -2.51205,
            1.00526, -1.01683, 1.70691, -1.86564, 1.84912,
            -1.33120, 2.35105, -0.45579, -0.57773, -0.55226,
        }
```
0.88371, 0.23138, 0.59984, 0.31971, 0.59849,
0.41873, -0.46955, 0.53003, -1.17203, 1.52937,
-0.48017, -0.93830, 1.00651, -0.92214, -0.42188,
-0.67010, 0.58079, -0.96193, 0.22763, -0.92214,
1.35697, -1.47008, 2.47841, -1.50522, 0.41650,
-0.21669, -0.90297, 0.00274, -1.04863, 0.66192,
-0.39143, 0.40779, -0.68174, -0.04700, -0.84469,
0.30735, -0.68412, 0.25888, -1.08642, 0.52928,
0.72168, -0.18199, -0.09499, 0.67610, 0.14636,
0.46846, -0.13989, 0.50856, -0.22268, 0.92756,
0.73069, 0.78998, -1.01650, 1.25637, -2.36179,
1.99616, -1.54326, 1.38220, 0.19674, -0.85241,
0.40463, 0.39523, -0.60721, 0.25041, -1.24967,
0.26727, 1.40042, -0.66963, 1.26049, -0.92074,
0.05909, -0.61926, 1.41550, 0.25537, -0.13240,
-0.07543, 0.10413, 1.42445, -1.37379, 0.44382,
-1.57210, 2.04702, -2.22450, 1.27698, 0.01073,
-0.88459, 0.88194, -0.25019, 0.70224, -0.41855,
0.93850, 0.36007, -0.46043, 0.18645, 0.06337,
0.29414, -0.20054, 0.83078, -1.62530, 2.64925,
-1.25355, 1.59094, -1.00684, 1.03196, -1.58045,
2.04295, -2.38264, 1.65095, -0.33273, -1.29092,
0.14020, -0.11434, 0.04392, 0.05293, -0.42277,
0.59143, -0.03347, -0.58457, 0.87030, 0.19985,
-0.73500, 0.73640, 0.29531, 0.22325, -0.60035,
1.42253, -1.11278, 1.30468, -0.41923, -0.38019,
0.50937, 0.23051, 0.46496, 0.02459, -0.68478,
0.25821, 1.17655, -2.26629, 1.41173, -0.68331};

ARMAEstimateMissing estMiss = null;
ARAutoUnivariate arAuto = null;
String title = " ";
String[] colLabels = new
String[]{"TIME", "ACTUAL", "PREDICTED", "DIFFERENCE"};
PrintMatrixFormat pmf = new PrintMatrixFormat();
PrintMatrix pm = new PrintMatrix();
pmf.NumberFormat = "0.000";
pmf.SetColumnLabels(colLabels);
pmf.FirstRowNumber = 1;
pm.SetColumnSpacing(3);

/* setup missing data arrays */
tpointsComplete = new int[200];
tpointsMiss = new int[194];
yMiss = new double[194];
for (i = 1; i <= 200; i++)
    tpointsComplete[i - 1] = i;
tpointsMiss[0] = tpointsComplete[0];
yMiss[0] = yComplete[0];
k = 0;
for (i = 1; i < 200; i++)
    }
/* Generate series with missing values */
if (i != 129 && i != 139 && i != 140 && i != 159 && i != 174 && i != 175)
k += 1;
tpointsMiss[k] = tpointsComplete[i];
yMiss[k] = yComplete[i];
}
}
n_obs = k + 1;
{
estMiss = new ARMAEstimateMissing(tpointsMiss, yMiss);
estMiss.MissingValueMethod = j;
switch (j)
{
    case ARMAEstimateMissing.MissingValueEstimation.Median:
        title = "MEDIAN ESTIMATES";
        break;
    case ARMAEstimateMissing.MissingValueEstimation.CubicSpline:
        title = "CUBIC SPLINE ESTIMATES";
        break;
    case ARMAEstimateMissing.MissingValueEstimation.AR_1:
        title = "AR(1) ESTIMATES";
        break;
    case ARMAEstimateMissing.MissingValueEstimation.AR_p:
        estMiss.Maxlag = maxlag;
title = "AR(P) ESTIMATES";
        break;
}
/* For some data it is useful to turn off warnings produced by
* the ARMA estimation process. This is only necessary for
* the AR_1 and AR_P estimation methods
*/
WarningObject w = Warning.WarningObject;
Warning.WarningObject = null;
y = estMiss.GetCompleteTimeSeries();
Warning.WarningObject = w; // turn on warnings
missing_index = estMiss.GetMissingTimes();
n_miss = y.Length - yMiss.Length;
double[][] printOutput = new double[n_miss][];
for (int i2 = 0; i2 < n_miss; i2++)
{
    printOutput[i2] = new double[4];
}
sum = 0;
for (i = 0; i < n_miss; i++)
{
    k = missing_index[i];
    printOutput[i][0] = tpointsComplete[k];
    printOutput[i][1] = tpointsMiss[k];
    printOutput[i][2] = yComplete[i];
    printOutput[i][3] = yMiss[k];
}
printOutput[i][1] = yComplete[k];
printOutput[i][2] = y[k];
printOutput[i][3] = Math.Abs(yComplete[k] - y[k]);
sum += Math.Pow(printOutput[i][3], 2);
}
arAuto = new ARAutoUnivariate(maxlag, y);
arAuto.Compute();
variance = arAuto.InnovationVariance;
pm.SetTitle(title);
pm.Print(pmf, printOutput);
missVar = sum / n_miss;
Console.Out.WriteLine;
{"Innovation Variance Analysis - Estimate (percent of actual)"};
Console.Out.WriteLine(" Missing Values(only): " + missVar + "%");
Console.Out.WriteLine(" Entire Series: " + variance + "%");
}
}

Output

<table>
<thead>
<tr>
<th>TIME</th>
<th>ACTUAL</th>
<th>PREDICTED</th>
<th>DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>130.00</td>
<td>-0.921</td>
<td>0.261</td>
</tr>
<tr>
<td>2</td>
<td>140.00</td>
<td>0.444</td>
<td>0.057</td>
</tr>
<tr>
<td>3</td>
<td>141.00</td>
<td>-1.572</td>
<td>0.057</td>
</tr>
<tr>
<td>4</td>
<td>160.00</td>
<td>2.649</td>
<td>0.047</td>
</tr>
<tr>
<td>5</td>
<td>175.00</td>
<td>-0.423</td>
<td>0.048</td>
</tr>
<tr>
<td>6</td>
<td>176.00</td>
<td>0.591</td>
<td>0.048</td>
</tr>
</tbody>
</table>

Innovation Variance Analysis - Estimate (percent of actual)
Missing Values(only): 1.91525937619167 (376%)
Entire Series: 0.535039841503757 (105%)

CUBIC SPLINE ESTIMATES

<table>
<thead>
<tr>
<th>TIME</th>
<th>ACTUAL</th>
<th>PREDICTED</th>
<th>DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>130.00</td>
<td>-0.921</td>
<td>1.541</td>
</tr>
<tr>
<td>2</td>
<td>140.00</td>
<td>0.444</td>
<td>-0.407</td>
</tr>
<tr>
<td>3</td>
<td>141.00</td>
<td>-1.572</td>
<td>2.497</td>
</tr>
<tr>
<td>4</td>
<td>160.00</td>
<td>2.649</td>
<td>-2.947</td>
</tr>
<tr>
<td>5</td>
<td>175.00</td>
<td>-0.423</td>
<td>0.251</td>
</tr>
<tr>
<td>6</td>
<td>176.00</td>
<td>0.591</td>
<td>0.380</td>
</tr>
</tbody>
</table>

Innovation Variance Analysis - Estimate (percent of actual)
Missing Values(only): 9.19346459339994 (1803%)
Entire Series: 0.75913799041326 (149%)

AR(1) ESTIMATES

<table>
<thead>
<tr>
<th>TIME</th>
<th>ACTUAL</th>
<th>PREDICTED</th>
<th>DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>130.00</td>
<td>-0.921</td>
<td>-0.930</td>
</tr>
<tr>
<td>2</td>
<td>140.00</td>
<td>0.444</td>
<td>1.028</td>
</tr>
<tr>
<td>3</td>
<td>141.00</td>
<td>-1.572</td>
<td>-0.745</td>
</tr>
<tr>
<td>4</td>
<td>160.00</td>
<td>2.649</td>
<td>1.229</td>
</tr>
</tbody>
</table>
**ARMAEstimateMissing.MissingValueEstimation Enumeration**

public enumeration Imsl.Stat.ARMAMissingValueEstimation

Missing value estimation methods.

**Fields**

**AR_1**

public Imsl.Stat.ARMAMissingValueEstimation AR_1

**Description**

Indicates that missing values should be estimated using an autoregressive time series with 1 lag.

**AR_p**

public Imsl.Stat.ARMAMissingValueEstimation AR_p

**Description**

Indicates that missing values should be estimated using an autoregressive time series with a maximum lag of Maxlag. By default Maxlag=10, but this can be changed using the Maxlag property.

**CubicSpline**

public Imsl.Stat.ARMAMissingValueEstimation CubicSpline

---

<table>
<thead>
<tr>
<th>TIME</th>
<th>ACTUAL</th>
<th>PREDICTED</th>
<th>DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>130.000</td>
<td>-0.921</td>
<td>0.032</td>
</tr>
<tr>
<td>2</td>
<td>140.000</td>
<td>0.444</td>
<td>0.565</td>
</tr>
<tr>
<td>3</td>
<td>141.000</td>
<td>-1.572</td>
<td>0.884</td>
</tr>
<tr>
<td>4</td>
<td>160.000</td>
<td>2.649</td>
<td>1.439</td>
</tr>
<tr>
<td>5</td>
<td>175.000</td>
<td>-0.423</td>
<td>0.421</td>
</tr>
<tr>
<td>6</td>
<td>176.000</td>
<td>0.591</td>
<td>0.553</td>
</tr>
</tbody>
</table>
## Description
Indicates that missing values should be estimated using cubic spline interpolation.

## Median

## Description
Indicates that missing values should be estimated using the median of the values just before and after the missing value gap.

## ARMAMaxLikelihood Class

`public class Imsl.Stat.ARMAMaxLikelihood`

Computes maximum likelihood estimates of parameters for an ARMA model with $p$ and $q$ autoregressive and moving average terms respectively.

ARMAMaxLikelihood computes estimates of parameters for a nonseasonal ARMA model given a sample of observations, $W_t$, for $t = 1, 2, \ldots, n$, where $n = z.Length$. The class is derived from the maximum likelihood estimation algorithm described by Akaike, Kitagawa, Arahata and Tada (1979), and the XSARMA routine published in the TIMSAC-78 Library.

The stationary time series $W_t$ with mean $\mu$ can be represented by the nonseasonal autoregressive moving average (ARMA) model by the following relationship:

$$
\phi(B)(W_t - \mu) = \theta(B)\alpha_t
$$

where

$t \in \mathbb{Z}, \ Z = \{..., -2, -1, 0, 1, 2, ...\}$,

$B$ is the backward shift operator defined by $B^k W_t = W_{t-k}$,

$$
\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p, \ p \geq 0.
$$

and

$$
\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \ldots - \theta_q B^q, \ q \geq 0.
$$

The ARMAMaxLikelihood class estimates the AR coefficients $\phi_1, \phi_2, \ldots, \phi_p$ and the MA coefficients $\theta_1, \theta_2, \ldots, \theta_q$ using maximum likelihood estimation.

ARMAMaxLikelihood checks the initial estimates for both the autoregressive and moving average coefficients to ensure that they represent a stationary and invertible series respectively.

If

$$
\phi_1, \phi_2, \ldots, \phi_p
$$

are the initial estimates for a stationary series then all (complex) roots of the following polynomial will fall outside the unit circle:

$$
1 - \phi_1 z - \phi_2 z^2 - \ldots - \phi_p z^p.
$$
If \( \theta_1, \theta_2, \ldots, \theta_q \) are initial estimates for an invertible series then all (complex) roots of the polynomial
\[
1 - \theta_1 z - \theta_2 z^2 - \ldots - \theta_q z^q
\]
will fall outside the unit circle.

By default, the order of the lags for the autoregressive terms is \( 1, 2, \ldots, p \) and \( 1, 2, \ldots, q \) for the moving average terms. However, this cannot be overridden.

Initial values for the AR and MA coefficients can be supplied via the SetAR and SetMA methods. Otherwise, initial estimates are computed internally by the method of moments. The class computes the roots of the associated polynomials. If the AR estimates represent a nonstationary series, ARMAMaxLikelihood issues a warning message and replaces the initial AR estimates with initial values that are stationary. If the MA estimates represent a noninvertible series, a terminal error is issued and new initial values must be sought.

ARMAMaxLikelihood also validates the final estimates of the AR coefficients to ensure that they too represent a stationary series. This is done to guard against the possibility that the internal log-likelihood optimizer converged to a nonstationary solution. If nonstationary estimates are encountered, an exception will be thrown.

For model selection, the ARMA model with the minimum value for AIC might be preferred,
\[
AIC = -2 \ln(L) + 2(p + q),
\]
where \( L \) is the value of the maximum likelihood function evaluated at the parameter estimates.

ARMAMaxLikelihood can also handle white noise processes, i.e. ARMA\((0,0)\) processes.

Properties

**BackwardOrigin**

```csharp
public int BackwardOrigin { get; set; }
```

**Description**
The maximum backward origin.

**Property Value**
An int scalar containing the maximum backward origin used in forecasting. BackwardOrigin must be greater than or equal to 0 and less than or equal to \( z.Length - \text{Math.Max}(p, q) \).

**Remarks**
By default, BackwardOrigin = 0.

**Confidence**

```csharp
public double Confidence { get; set; }
```
Description
The confidence level for calculating confidence limit deviations returned from GetDeviations.

Property Value
A double scalar specifying the confidence level used in computing forecast confidence intervals.

Remarks
The confidence limits must be greater than 0.0 and less than 1.0. Suggested confidence limits are 0.90, 0.95, and 0.99. By default, Confidence = 0.95.

Constant
public double Constant {get; set; }

Description
The constant parameter in the ARMA series.

Property Value
A double scalar containing the constant term in the ARMA model.

Remarks
By default, the constant term is initially estimated using ARMA method of moments estimation.

GradientTolerance
public double GradientTolerance {get; set; }

Description
The gradient tolerance for the convergence algorithm.

Property Value
A double scalar containing the tolerance used for numerically estimating the gradient by differences.

Remarks
By default, GradientTolerance = 1e-04.

InnovationVariance
public double InnovationVariance {get; }

Description
The estimated innovation variance of this series.

Property Value
A double scalar containing the estimated innovation variance for the time series.

Likelihood
public double Likelihood {get; }

Description
The final estimate for \(-2\ln(L)\), where \(L\) is equal to the likelihood function evaluated using the final parameter estimates.
Property Value

A double scalar equal to the log likelihood function, \(-2\ln(L)\).

Exceptions

- `Imsl.Stat.NonStationaryException` is thrown if the final maximum likelihood estimates for the time series are nonstationary.
- `Imsl.Stat.NonInvertibleException` is thrown if the final maximum likelihood estimates for the time series are noninvertible.
- `Imsl.Stat.InitialMAException` is thrown if the initial values provided for the moving average terms using `SetMA` are noninvertible. In this case, `ARMAMaxLikelihood` terminates and does not compute the time series estimates.

MaxIterations

```java
public int MaxIterations {get; set; }
```

Description

The maximum number of iterations.

Property Value

An int scalar containing the maximum number of iterations allowed in the nonlinear equation solver used in both the method of moments and least-squares algorithms.

Remarks

By default, `MaxIterations = 300`.

Mean

```java
public double Mean {get; set; }
```

Description

The mean used for centering the series.

Property Value

A double scalar containing the value for centering the time series. By default the series is not centered.

P

```java
public int P {get; }
```

Description

The number of autoregressive terms in the ARMA model.

Property Value

An int scalar value of \(p\), containing the number of autoregressive terms in the ARMA model.

Q

```java
public int Q {get; }
```

Description

The number of moving average terms in the ARMA model.
Property Value
An int scalar value of q, containing the number of moving average terms in the ARMA model.

Tolerance
public double Tolerance {get; set; }

Description
The tolerance for the convergence algorithm.

Property Value
A double scalar containing the value to use for the convergence tolerance during maximum likelihood estimation.

Remarks
By default, Tolerance = 2.220446049e-016.

Constructor

ARMAMaxLikelihood
public ARMAMaxLikelihood(int p, int q, double[] z)

Description
Constructor for ARMAMaxLikelihood.

Parameters
p – An int scalar equal to the number of autoregressive (AR) parameters.
q – A int scalar equal to the number of moving average (MA) parameters.
z – A double array containing the time series.

Exceptions
Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.
Methods

Compute
public void Compute()

Description
Computes the exact maximum likelihood estimates for the autoregressive and moving average parameters of an ARMA time series.

Exceptions
- `Imsl.Stat.NonStationaryException` is thrown if the final maximum likelihood estimates for the time series are nonstationary.
- `Imsl.Stat.NonInvertibleException` is thrown if the final maximum likelihood estimates for the time series are noninvertible.
- `Imsl.Stat.InitialMAException` is thrown if the initial values provided for the moving average terms using `SetMA` are noninvertible. In this case, `ARMAMaxLikelihood` terminates and does not compute the time series estimates.

Forecast
public double[,] Forecast(int nForecast)

Description
Returns forecasts for lead times \( l = 1, 2, \ldots, n\text{Forecast} \) at origins \( z.\text{Length}\text{-BackwardOrigin}\text{-1+j} \) where \( j = 1, \ldots, \text{BackwardOrigin}\text{+1} \).

Parameter
- `nForecast` – An int scalar equal to the number of requested forecasts.

Returns
A double matrix of dimensions of \( n\text{Forecast} \) by \( \text{BackwardOrigin} \text{+1} \) containing the forecasts. The forecasts are for lead times \( l = 1, 2, \ldots, n\text{Forecast} \) at origins \( z.\text{Length}\text{-BackwardOrigin}\text{-1+j} \) where \( j = 1, \ldots, \text{BackwardOrigin}\text{+1} \).

Exceptions
- `Imsl.Stat.NonStationaryException` is thrown if the final maximum likelihood estimates for the time series are nonstationary.
- `Imsl.Stat.NonInvertibleException` is thrown if the final maximum likelihood estimates for the time series are noninvertible.
- `Imsl.Stat.InitialMAException` is thrown if the initial values provided for the moving average terms using `SetMA` are noninvertible. In this case, `ARMAMaxLikelihood` terminates and does not compute the time series estimates.
- `Imsl.Stat.MatrixSingularException` is thrown if the input matrix is singular.
- `Imsl.Stat.TooManyCallsException` is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.

---

**GetAR**

```csharp
public double[] GetAR()
```

**Description**

Returns the final autoregressive parameter estimates.

**Returns**

A `double` array of length `p` containing the final autoregressive parameter estimates.

---

**GetDeviations**

```csharp
public double[] GetDeviations()
```

**Description**

Returns the deviations for each forecast used for calculating the forecast confidence limits.

**Returns**

A `double` array of length `nForecast+BackwardOrigin` containing the deviations for calculating forecast confidence intervals. The confidence level is specified in `Confidence`.

**Remarks**

Note that the `Forecast` or `GetForecast` method must be invoked first before invoking this method. Otherwise, the method returns `null`.

---

**GetForecast**

```csharp
public double[] GetForecast(int nForecast)
```

**Description**

Returns forecasts

**Parameter**

- `nForecast` – An input `int` representing the number of requested forecasts beyond the last value in the series.

**Returns**

A `double` array containing the `nForecast+BackwardOrigin` forecasts. The first `BackwardOrigin` forecasts are one-step ahead forecasts for the last `BackwardOrigin` values in the series. The next `nForecast` values in the returned series are forecasts for the next values beyond the series.
Exceptions

**Imsl.Stat.NonStationaryException** is thrown if the final maximum likelihood estimates for the time series are nonstationary.

**Imsl.Stat.NonInvertibleException** is thrown if the final maximum likelihood estimates for the time series are noninvertible.

**Imsl.Stat.InitialMAException** is thrown if the initial values provided for the moving average terms using `SetMA` are noninvertible. In this case, `ARMAMaxLikelihood` terminates and does not compute the time series estimates.

**Imsl.Stat.MatrixSingularException** is thrown if the input matrix is singular.

**Imsl.Stat.TooManyCallsException** is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.

**Imsl.Stat.IncreaseErrRelException** is thrown if the bound for the relative error is too small.

**Imsl.Stat.NewInitialGuessException** is thrown if the iteration has not made good progress.

**Imsl.Stat.IllConditionedException** is thrown if the problem is ill-conditioned.

**Imsl.Stat.TooManyIterationsException** is thrown if the maximum number of iterations is exceeded.

**Imsl.Stat.TooManyFunctionEvaluationsException** is thrown if the maximum number of function evaluations is exceeded.

**Imsl.Stat.TooManyJacobianEvalException** is thrown if the maximum number of Jacobian evaluations is exceeded.

---

**GetGradients**

```java
public double[] GetGradients()
```

**Description**

Returns the gradients for the final parameter estimates.

**Returns**

A `double` array of length `p+q` containing the gradients of the final parameter estimates.

---

**GetMA**

```java
public double[] GetMA()
```

**Description**

Returns the final moving average parameter estimates.

**Returns**

A `double` array of length `q` containing the final moving average parameter estimates.

---

**GetPsiWeights**

```java
public double[] GetPsiWeights()
```
Description
Returns the psi weights used for calculating forecasts from the infinite order moving average form of the ARMA model.

Returns
A double array of length nForecast containing the psi weights of the infinite order moving average form of the model.

GetResiduals
public double[] GetResiduals()

Description
The current values of the vector of residuals.

Returns
A double array of length BackwardOrigin containing the residuals for the last BackwardOrigin values in the time series. The Compute and either the Forecast or GetForecast methods must be called before calling this method.

GetTimeSeries
public double[] GetTimeSeries()

Description
Returns the time series used to construct ARMAMaxLikelihood.

Returns
A double containing the values of the time series passed to the class constructor.

IsInvertible
public bool IsInvertible(double[] ma)

Description
Tests whether the coefficients in ma are invertible.

Parameter
ma – A double array containing the coefficients for the moving average terms in an ARMA model.

Returns
A bool scalar equal to true if the coefficients in ma are invertible and false otherwise.

IsStationary
public bool IsStationary(double[] ar)

Description
Tests whether the coefficients in ar are stationary.

Parameter
ar – A double array containing the coefficients for the autoregressive terms in an ARMA model.
Returns
A bool scalar equal to true if the coefficients in ar are stationary and false otherwise.

SetAR
public void SetAR(double[] ar)

Description
Sets the initial values for the autoregressive terms to the p values in ar.

Parameter
ar – An input double array of length p containing the initial values for the autoregressive terms. If this method is not called, initial values are computed by method of moments in the ARMA class.

SetMA
public void SetMA(double[] ma)

Description
Sets the initial values for the moving average terms to the q values in ma.

Parameter
ma – A double array of length q containing the initial values for the moving average terms. If this method is not called, initial values are computed by method of moments in the ARMA class.

Example: ARMAMaxLikelihood
Consider the Wolfer Sunspot Data (Anderson 1971, p. 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869. The maximum likelihood estimates \( \hat{\theta}_0, \hat{\phi}_1, \hat{\phi}_2, \) and \( \hat{\theta}_1 \) are computed for the ARMA(2, 1) model

\[
z_t = \theta_0 + \phi_1 z_{t-1} + \phi_2 z_{t-2} - \theta_1 A_{t-1} + A_t
\]

where the errors \( A_t \) are independently normally distributed with mean zero and variance \( \sigma_A^2 \). The maximum likelihood estimates from ARMAMaxLikelihood are compared to the same estimates using the method of moments and least squares from the ARMA class. For each method, the coefficients and forecasts for the last ten years, 1860-1869, are compared. The method of moments and maximum likelihood estimates produced similar results. However, the least squares method does not converge using the default relative error convergence criteria. As a result, a warning is produced and the estimates are very different from the method of moments and maximum likelihood estimates.

```java
using System;
using Imsl.Stat;
using Imsl.Math;
public class ARMAMaxLikelihoodEx1 {
```
public static void Main(String[] args) {
    int backwardOrigin = 0, n_forecast = 10, n_series = 100;
    double[] sunspots = {
        100.8, 81.6, 66.5, 34.8, 30.6, 7, 19.8, 92.5,
        154.4, 125.9, 84.8, 68.1, 38.5, 22.8, 10.2,
        24.1, 82.9, 132, 130.9, 118.1, 89.9, 66.6,
        60, 46.9, 41, 21.3, 16, 6.4, 4.1, 6.8, 14.5,
        34, 45, 43.1, 47.5, 42.2, 28.1, 10.1, 8.1,
        2.5, 0, 1.4, 5, 12.2, 13.9, 35.4, 45.8, 41.1,
        30.4, 23.9, 15.7, 6.6, 4, 1.8, 8.5, 16.6,
        36.3, 49.7, 62.5, 67, 71, 47.8, 27.5, 8.5,
        13.2, 56.9, 121.5, 138.3, 103.2, 85.8, 63.2,
        36.8, 24.2, 10.7, 15, 40.1, 61.5, 98.5, 124.3,
        95.9, 66.5, 64.5, 54.2, 39, 20.6, 6.7, 4.3,
        22.8, 54.8, 93.8, 95.7, 77.2, 59.1, 44.7,
        30.5, 16.3, 7.3, 37.3, 73.9
    };
    double[] z = null;
    double[] arMM, maMM;
    double constantMM;
    double[] arLS, maLS;
    double constantLS;
    double[] ar, ma;
    double constant;
    double[,] forecastMM, forecastLS, forecast;
    double[] deviations;
    double likelihood, var, varMM, varLS;
    double[] avgDev = new double[0, 0, 0.0];
    ARMA armaMM = null;
    ARMA armaLS = null;
    ARMA MaxLikelihood maxArma = null;
    double[,] printOutput = null;
    double[,] printOutput2 = null;
    String[] colLabels = {"Method of Moments",
        "Least Squares",
        "Maximum Likelihood"};
    String[] colLabels1 = {"Least Squares", "Maximum Likelihood"};
    String[] colLabels2 = {"Observed Sunspots", "Method of Moments",
        "Least Squares", "Maximum Likelihood"};
    String[] colLabels3 = {"Lower Confidence Limit", "Forecast",
        "Upper Confidence Limit"};
    PrintMatrixFormat pmf = new PrintMatrixFormat();
    PrintMatrix pm = new PrintMatrix();
    pm.SetColumnSpacing(3);
    pmf.SetNoRowLabels();
    pmf.SetColumnLabels(colLabels);
    printOutput = new double[1, 3];
    printOutput2 = new double[n_forecast, 4];
    z = new double[n_series];
    for (int i = 0; i < n_series; i++)
        z[i] = sunspots[i];
    /* Method of Moments ARMA(2,1) Estimation */
    armaMM = new ARMA(2, 1, z);
    armaMM.Compute();
    armaMM.BackwardOrigin = backwardOrigin;
arMM = armaMM.GetAR();
amMM = armaMM.GetMA();
constantMM = armaMM.Constant;
forecastMM = armaMM.Forecast(n_forecast);
varMM = armaMM.InnovationVariance;

/* Least Squares ARMA(2,1) Estimation */
armaLS = new ARMA(2, 1, z);
armaLS.Method = Imsl.Stat.ARMAMaxLikelihood;LS_ParamEstimation.LeastSquares;
armaLS.Compute();
arLS = armaLS.GetAR();
maLS = armaLS.GetMA();
constantLS = armaLS.Constant;
varLS = armaLS.InnovationVariance;
forecastLS = armaLS.Forecast(n_forecast);

/* Maximum Likelihood ARMA(2,1) Estimation */
maxArma = new ARMAMaxLikelihood(2, 1, z);
maxArma.Compute();
ar = maxArma.GetAR();
ma = maxArma.GetMA();
constant = maxArma.Constant;
likelihood = maxArma.Likelihood;
var = maxArma.InnovationVariance;
maxArma.Confidence = 0.9;
forecast = maxArma.Forecast(n_forecast);
deviations = maxArma.GetDeviations();

printOutput[0,0] = constantMM;
printOutput[0,1] = constantLS;
printOutput[0,2] = constant;
pm.SetTitle("ARMA(2,1) - Constant Term");
pm.Print(pmf, printOutput);
printOutput[0,0] = arMM[0];
printOutput[0,1] = arLS[0];
printOutput[0,2] = ar[0];
pm.SetTitle("ARMA(2,1) - AR(1) Coefficient");
pm.Print(pmf, printOutput);
printOutput[0,0] = arMM[1];
printOutput[0,1] = arLS[1];
printOutput[0,2] = ar[1];
pm.SetTitle("ARMA(2,1) - AR(2) Coefficient");
pm.Print(pmf, printOutput);
printOutput[0,0] = maMM[0];
printOutput[0,1] = maLS[0];
printOutput[0,2] = ma[0];
pm.SetTitle("ARMA(2,1) - MA(1) Coefficient");
pm.Print(pmf, printOutput);

Console.Out.WriteLine("INNOVATION VARIANCE:");
Console.Out.WriteLine("Method of Moments " + varMM);
Console.Out.WriteLine("Least Squares " + varLS);
Console.Out.WriteLine("Maximum Likelihood " + var);

Time Series and Forecasting
for (int i = 0; i < n_forecast; i++)
{
    printOutput2[i,0] = sunspots[90 + i];
    printOutput2[i,1] = forecastMM[i,backwardOrigin];
    printOutput2[i,2] = forecastLS[i,backwardOrigin];
    printOutput2[i,3] = forecast[i,backwardOrigin];
}

pm.SetTitle("SUNSPOT FORECASTS FOR 1860-1869");
pmf.SetColumnLabels(colLabels2);
pmf.FirstRowNumber = 1860;
pm.Print(pmf, printOutput2);

/* Get Confidence Interval Deviations */
printOutput2 = new double[n_forecast,3];
for (int i = 0; i < n_forecast; i++)
{
    printOutput2[i,0] = Math.Max(0, forecast[i,backwardOrigin] - deviations[i + backwardOrigin]);
    printOutput2[i,1] = forecast[i,backwardOrigin];
    printOutput2[i,2] = forecast[i,backwardOrigin] + deviations[i];
}

pmf.SetColumnLabels(colLabels3);
pmf.FirstRowNumber = 1860;
pm.SetTitle("SUNSPOT MAX. LIKELIHOOD 90% CONFIDENCE INTERVALS");
pm.Print(pmf, printOutput2);

Output

ARMA(2,1) - Constant Term

Method of Moments  Least Squares  Maximum Likelihood
15.5439819435637   17.9316722242554   15.7580039183081

ARMA(2,1) - AR(1) Coefficient

Method of Moments  Least Squares  Maximum Likelihood
1.2442577984372    1.53128221858841   1.22506595122835

ARMA(2,1) - AR(2) Coefficient

Method of Moments  Least Squares  Maximum Likelihood
-0.575149766040151  -0.894433318954122  -0.56051392292258

ARMA(2,1) - MA(1) Coefficient

Method of Moments  Least Squares  Maximum Likelihood
-0.124089747872598  -0.132018760615022  -0.382891621968495

INNOVATION VARIANCE:

Method of Moments  287.24203738122
Least Squares  239.68797223859
Maximum Likelihood  214.50878425629

SUNSPOT FORECASTS FOR 1860-1869

<table>
<thead>
<tr>
<th>Year</th>
<th>Observed Sunspots</th>
<th>Method of Moments</th>
<th>Least Squares</th>
<th>Maximum Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1860</td>
<td>95.7</td>
<td>87.6861374610048</td>
<td>98.0265069437306</td>
<td>87.9339016395945</td>
</tr>
<tr>
<td>1861</td>
<td>77.2</td>
<td>82.1446177467774</td>
<td>101.939296986813</td>
<td>82.0608538716695</td>
</tr>
<tr>
<td>1862</td>
<td>59.1</td>
<td>67.3203794962267</td>
<td>86.351331124405</td>
<td>66.999785793059</td>
</tr>
<tr>
<td>Year</td>
<td>Sunspot Max.</td>
<td>Forecast</td>
<td>Upper Confidence Limit</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>----------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>1860</td>
<td>63.8431804305709</td>
<td>87.9339016395945</td>
<td>112.024622848618</td>
<td></td>
</tr>
<tr>
<td>1861</td>
<td>36.4438944595113</td>
<td>82.0608538716595</td>
<td>127.677813238308</td>
<td></td>
</tr>
<tr>
<td>1862</td>
<td>10.134623371631</td>
<td>66.99975755390359</td>
<td>123.864948146981</td>
<td></td>
</tr>
<tr>
<td>1863</td>
<td>0</td>
<td>51.8409090696475</td>
<td>112.081240519376</td>
<td></td>
</tr>
<tr>
<td>1864</td>
<td>0</td>
<td>41.7122237493372</td>
<td>102.18743849516</td>
<td></td>
</tr>
<tr>
<td>1865</td>
<td>0</td>
<td>37.8006776731389</td>
<td>98.4521257842839</td>
<td></td>
</tr>
<tr>
<td>1866</td>
<td>0</td>
<td>38.1086417046268</td>
<td>99.9504700070846</td>
<td></td>
</tr>
<tr>
<td>1867</td>
<td>0</td>
<td>41.469854513895</td>
<td>103.74778523107</td>
<td></td>
</tr>
<tr>
<td>1868</td>
<td>0</td>
<td>45.2249946909406</td>
<td>107.46440111018</td>
<td></td>
</tr>
<tr>
<td>1869</td>
<td>0</td>
<td>47.945927562368</td>
<td>109.958371498445</td>
<td></td>
</tr>
</tbody>
</table>

Imsl.Stat.ARMA: Relative function convergence - Both the scaled actual and predicted reductions in the function are less than or equal to the relative function convergence tolerance "ConvergenceTolerance" = 5.00000041370186E-10.

Imsl.Stat.ARMA: Least squares estimation of the parameters has failed to converge. Increase "length" and/or "tolerance" and/or "convergence_tolerance". The estimates of the parameters at the last iteration may be used as new starting values.

**ARMAOutlierIdentification Class**

public class Imsl.Stat.ARMAOutlierIdentification

Dects and determines outliers and simultaneously estimates the model parameters in a time series whose underlying outlier free series follows a general seasonal or nonseasonal ARMA model. This class also allows computation of forecasts.

Consider a univariate time series \( \{Y_t\} \) that can be described by the following multiplicative seasonal ARIMA model of order \((p,0,q) \times (0,d,0)_s\):

\[
Y_t - \mu = \frac{\theta(B)}{\Delta^d \phi(B)} a_t, \quad t = 1, \ldots, n
\]

Here, \( \Delta^d = (1 - B)^d \), \( \theta(B) = 1 - \theta_1 B - \ldots - \theta_q B^q \), \( \phi(B) = 1 - \phi_1 B - \ldots - \phi_p B^p \). \( B \) is the lag operator, \( B^k Y_t = Y_{t-k} \). \( \{a_t\} \) is a white noise process, and \( \mu \) denotes the mean of the series \( \{Y_t\} \).

**Outlier detection and parameter estimation**

In general, \( \{Y_t\} \) is not directly observable due to the influence of outliers. Chen and Liu (1993) distinguish between four types of outliers: innovational outliers (IO), additive outliers (AO), temporary
changes (TC) and level shifts (LS). If an outlier occurs as the last observation of the series, then Chen and Liu’s algorithm is unable to determine the outlier’s classification. In class ARMAOutlierIdentification, such an outlier is called a UI (unable to identify) and is treated as an innovational outlier.

In order to take the effects of multiple outliers occurring at time points $t_1, \ldots, t_m$ into account, Chen and Liu consider the following model:

$$Y_t^* - \mu = \sum_{j=1}^{m} \omega_j L_j(B)L_t(t_j) + \theta(B) \Delta \phi(B) \alpha_t$$

Here, $\{Y_t^*\}$ is the observed outlier contaminated series, and $\omega_j$ and $L_j(B)$ denote the magnitude and dynamic pattern of outlier $j$, respectively. $L_t(t_j)$ is an indicator function that determines the temporal course of the outlier effect, $L_t(t_j) = 1$, $L_t(t_j) = 0$ otherwise. Note that $L_j(B)$ operates on $L_t$ via $B^k L_t = L_{t-k}, k = 0, 1, \ldots$.

The last formula shows that the outlier free series $\{Y_t\}$ can be obtained from the original series $\{Y_t^*\}$ by removing all occurring outlier effects:

$$Y_t = Y_t^* - \sum_{j=1}^{m} \omega_j L_j(B)L_t(t_j)$$

The different types of outliers are characterized by different values for $L_j(B)$:

1. $L_j(B) = \frac{\theta(B)}{\Delta \phi(B)}$ for an innovational outlier,
2. $L_j(B) = 1$ for an additive outlier,
3. $L_j(B) = (1 - B)^{-1}$ for a level shift outlier and
4. $L_j(B) = (1 - \delta B)^{-1}, 0 < \delta < 1$, for a temporary change outlier.

Class ARMAOutlierIdentification is an implementation of Chen and Liu’s algorithm. It determines the coefficients in $\phi(B)$ and $\theta(B)$ and the outlier effects in the model for the observed series jointly in three stages. The magnitude of the outlier effects is determined by least squares estimates. Outlier detection itself is realized by examination of the maximum value of the standardized statistics of the outlier effects. For a detailed description, see Chen and Liu’s original paper (1993).

Intermediate and final estimates for the coefficients in $\phi(B)$ and $\theta(B)$ are computed by the Compute methods from classes ARMA and ARMAMaxLikelihood. If the roots of $\phi(B)$ or $\theta(B)$ lie on or within the unit circle, then the algorithm stops with an appropriate exception. In this case, different values for $p$ and $q$ should be tried.

**Forecasting**

From the relation between original and outlier free series,

$$Y_t^* = Y_t + \sum_{j=1}^{m} \omega_j L_j(B)L_t(t_j)$$
it follows that the Box-Jenkins forecast at origin $t$ for lead time $l$, $\hat{Y}_t^*(l)$, can be computed as

$$\hat{Y}_t^*(l) = \hat{Y}_t(l) + \sum_{j=1}^{m} \omega_j L_j(B)I_{t+j}(t), \quad l = 1, \ldots, n$$

Therefore, computation of the forecasts for \{Y_t^*\} is done in two steps:

1. Computation of the forecasts for the outlier free series \{Y_t\}.
2. Computation of the forecasts for the original series \{Y_t^*\} by adding the multiple outlier effects to the forecasts for \{Y_t\}.

**Step 1: Computation of the forecasts for the outlier free series \{Y_t\}**

Since

$$\varphi(B)(Y_t - \mu) = \theta(B)a_t$$

where

$$\varphi(B) := \Delta^d \phi(B) = 1 - \phi_1 B - \ldots - \phi_p B^p$$

the Box-Jenkins forecast at origin $t$ for lead time $l$, $\hat{Y}_t(l)$, can be computed recursively as

$$\hat{Y}_t(l) = (1 - \sum_{j=1}^{p+d} \varphi_j) \mu + \sum_{j=1}^{p+d} \varphi_j \hat{Y}_t(l-j) - \sum_{j=1}^{q} \theta_j a_{t+l-j}$$

Here,

$$\hat{Y}_t(l-j) = \begin{cases} Y_{t+l-j} & \text{for } \quad l-j \leq 0 \\ \hat{Y}_t(l-j) & \text{for } \quad l-j > 0 \end{cases}$$

and

$$a_k = \begin{cases} 0 & \text{for } \quad k \leq \max\{1, p+d\} \\ Y_k - \hat{Y}_{k-1}(1) & \text{for } \quad k = \max\{1, p+d\} + 1, \ldots, n \end{cases}$$

**Step 2: Computation of the forecasts for the original series \{Y_t^*\} by adding the multiple outlier effects to the forecasts for \{Y_t\}**

The formulas for $L_j(B)$ for the different types of outliers are as follows:

Innovational outlier (IO)

$$L_j(B) = \frac{\theta(B)}{\Delta^d \phi(B)} = \psi(B) = \sum_{k=0}^{\infty} \psi_k B^k, \quad \psi_0 = 1$$

Additive outliers (AO)

$$L_j(B) = 1$$

Level shifts (LS)

$$L_j(B) = \frac{1}{1-B} = \sum_{k=0}^{\infty} B^k$$

---

**Time Series and Forecasting**

[ARMAOutlierIdentification • 921]
Temporary changes (TC)

\[ L_j(B) = \frac{1}{1-\delta B} = \sum_{k=0}^{\infty} \delta^k B^k \]

Assuming the outlier occurs at time point \( t_j \), the outlier impact is therefore:

Innovational outliers (IO)

\[ \omega_j L_j(B) I_t(t_j) = \begin{cases} 0 & \text{for } t < t_j \\ \omega_j \psi_k & \text{for } t = t_j + k, k \geq 0 \end{cases} \]

Additive outliers (AO)

\[ \omega_j L_j(B) I_t(t_j) = \begin{cases} 0 & \text{for } t \neq t_j \\ \omega_j & \text{for } t = t_j \end{cases} \]

Level shifts (LS)

\[ \omega_j L_j(B) I_t(t_j) = \begin{cases} 0 & \text{for } t < t_j \\ \omega_j & \text{for } t = t_j + k, k \geq 0 \end{cases} \]

Temporary changes (TC)

\[ \omega_j L_j(B) I_t(t_j) = \begin{cases} 0 & \text{for } t < t_j \\ \omega_j \delta^k & \text{for } t = t_j + k, k \geq 0 \end{cases} \]

From these formulas, the forecasts \( \hat{Y}_t^*(l) \) can be computed easily. The 100(1 - \( \alpha \)) percent probability limits for \( Y_{t+l}^* \) and \( Y_{t+l} \) are given by

\[ \hat{Y}_t^*(l) \text{ (or } \hat{Y}_t(l), \text{ resp.) } \pm u_{\alpha/2} (1 + \sum_{j=1}^{l-1} \psi_j^2)^{1/2} s_a \]

where \( u_{\alpha/2} \) is the 100(1 - \( \alpha/2 \)) percentile of the standard normal distribution, \( s_a^2 \) is an estimate of the variance \( \sigma_a^2 \) of the random shocks, and the \( \psi \) weights \( \{ \psi_j \} \) are the coefficients in

\[ \psi(B) := \sum_{k=0}^{\infty} \psi_k B^k := \frac{\theta(B)}{\Delta^2 \phi(B)}, \psi_0 = 1. \]

For a detailed explanation of these concepts, see chapter 5: “Forecasting” in Box, Jenkins and Reinsel (1994).

**Fields**

**ADDITIVE**

```
public int ADDITIVE
```
Description
Indicates detection of an additive outlier.

INNOVATIONAL
public int INNOVATIONAL

Description
Indicates detection of an innovational outlier.

LEVEL SHIFT
public int LEVEL SHIFT

Description
Indicates detection of a level shift outlier.

TEMPORARY CHANGE
public int TEMPORARY CHANGE

Description
Indicates detection of a temporary change outlier.

UNABLE TO IDENTIFY
public int UNABLE TO IDENTIFY

Description
Indicates detection of an outlier that cannot be categorized.

Properties

AccuracyTolerance
public double AccuracyTolerance {get; set; }

Description
The tolerance value controlling the accuracy of the parameter estimates.

Property Value
A double scalar, a positive tolerance value controlling the accuracy of parameter estimates during outlier detection.
Default: AccuracyTolerance = 0.001.

AIC
public double AIC {get; }

Description
Returns Akaike’s information criterion (AIC).
Property Value
A double scalar containing Akaike's information criterion (AIC) for the outlier free series.

Remarks
The Compute method must be called before using this property. Otherwise, an InvalidOperationException exception is thrown.

AICC
public double AICC {get; }

Description
Returns Akaike's Corrected Information Criterion (AICC).

Property Value
A double scalar containing Akaike's Corrected Information Criterion (AICC) for the outlier free series.

Remarks
The Compute method must be called before using this property. Otherwise, an InvalidOperationException exception is thrown.

BIC
public double BIC {get; }

Description
Returns the Bayesian Information Criterion (BIC).

Property Value
A double scalar containing the Bayesian Information Criterion (BIC) for the outlier free series.

Remarks
The Compute method must be called before using this property. Otherwise, an InvalidOperationException exception is thrown.

Confidence
public double Confidence {get; set; }

Description
The confidence level for calculating confidence limit deviations via method GetDeviations.

Property Value
A double scalar specifying the confidence level used in computing forecast confidence intervals.
Default: Confidence = 0.95.

Remarks
The confidence levels must be greater than 0.0 and less than 1.0. Typical level choices are 0.90, 0.95, and 0.99.

Constant
public double Constant {get; }
Description
Returns the constant parameter estimate.

Remarks
The Compute method must be called before using this property. Otherwise, an
InvalidOperationException exception is thrown.

CriticalValue
public double CriticalValue {get; set; }

Description
The critical value used as a threshold during outlier detection.

Property Value
A double scalar, the critical value used as a threshold for the statistics used in the outlier detection.
Default: CriticalValue = 3.0.

Remarks
The critical value must be greater than zero.

Delta
public double Delta {get; set; }

Description
The dampening effect parameter.

Property Value
A double scalar, the dampening effect parameter used in the detection of a Temporary Change Outlier (TC).
Default: Delta = 0.

Remarks
The dampening effect parameter must be greater than 0 and less than 1.

NumberOfOutliers
public int NumberOfOutliers {get; }

Description
Returns the number of outliers detected.

Remarks
The Compute method must be called before using this property. Otherwise, an
InvalidOperationException exception is thrown.

RelativeError
public double RelativeError {get; set; }

Description
The stopping criterion for use in the nonlinear equation solver.
Property Value
A double positive scalar containing the stopping criterion for use in the nonlinear equation solver used in the least-squares algorithm.
Default: RelativeError=1.0e-10.

ResidualStandardError
public double ResidualStandardError {get; }

Description
Returns the residual standard error of the outlier free series.

Property Value
A double scalar containing the standard error of the outlier free series.

Remarks
The Compute method must be called before using this property. Otherwise, an InvalidOperationException exception is thrown.

Constructor

 ARMAOutlierIdentification
public ARMAOutlierIdentification(double[] z)

Description
Constructor for ARMAOutlierIdentification.

Parameter
z – A double array containing the observations.

Methods

 Compute
public void Compute(int[] model)

Description
Detects and determines outliers and simultaneously estimates the model parameters for the given time series.

Parameter
model – An int array of length 4 containing the numbers p, q, s, d of the ARIMA(p,0,q) x (0,d,0), model the outlier free series is following. It is required that p, q and d are non-negative and s is positive and consistent with z.Length.
Exceptions

Imsl.Stat.NonStationaryException is thrown if the intermediate or final maximum likelihood estimates for the time series are nonstationary.

Imsl.Stat.NonInvertibleException is thrown if the intermediate or final maximum likelihood estimates for the time series are noninvertible.

Imsl.Stat.InitialMAException is thrown if the initial values provided for the moving average terms are noninvertible. In this case, ARMAMaxLikelihood terminates and does not compute the time series estimates.

Imsl.Math.DidNotConvergeException is thrown if the algorithm computing the roots of the AR- or MA-polynomial does not converge.

Imsl.Stat.MatrixSingularException is thrown if one of the matrices used in classes ARMA or ARMAMaxLikelihood is singular.

Imsl.Stat.TooManyCallsException is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.

Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.

Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.

Imsl.Stat.IlleConditionedException is thrown if the problem is ill-conditioned.

Imsl.Stat.TooManyIterationsException is thrown if the maximum number of iterations is exceeded.

Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the maximum number of function evaluations is exceeded.

Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.

Imsl.Math.SingularMatrixException is thrown if during the computation of a small perturbation of the matrix product $A^T A$, it is found that $A$, the matrix used in the determination of the $\omega$ weights, is singular.

Imsl.Math.NotSPDEXception is thrown if during the computation of a small perturbation of the matrix product $A^T A$, it is found that $A$, the matrix used in the determination of the $\omega$ weights, is not positive definite.

ComputeForecasts

class public void ComputeForecasts(int nForecast)

Description

Computes forecasts, associated probability limits and $\psi$ weights for an outlier contaminated time series whose underlying outlier free series obeys a general seasonal or non-seasonal ARMA model.

Parameter

nForecast – An int scalar containing the maximum lead time for forecasts. nForecast must be greater than 0. Forecast origin is the time point of the last observed value in the time series, n. Forecasts are computed for lead times 1, 2, ..., nForecast, i.e. time points $n+1, n+2, ..., n+nForecast$. 

Time Series and Forecasting ARMAOutlierIdentification • 927
Remarks
The Compute method must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.

GetAR
public double[] GetAR()

Description
Returns the final autoregressive parameter estimates.

Returns
A double array of length \( p = \text{model}[0] \) containing the final autoregressive parameter estimates.

Remarks
The Compute method must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.

GetDeviations
public double[] GetDeviations()

Description
Returns the deviations used for calculating the forecast confidence limits.

Returns
A double array of length \( n\text{Forecast} \) containing the deviations from each forecast for calculating forecast confidence intervals.

Remarks
The confidence level is specified by the Confidence property. Method ComputeForecasts has to be invoked before this method is called. Otherwise, an InvalidOperationException exception is thrown.

GetForecast
public double[] GetForecast()

Description
Returns forecasts for the original outlier contaminated series.

Returns
A double array of length \( n\text{Forecast} \) containing the forecasts for the original series.

Remarks
Method ComputeForecasts must be invoked before this method is called. Otherwise, an InvalidOperationException exception is thrown.

GetMA
public double[] GetMA()

Description
Returns the final moving average parameter estimates.
Returns

A double array of length \( q = \text{model}[1] \) containing the final moving average parameter estimates.

Remarks

Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.

GetOmegaWeights

public double[] GetOmegaWeights()

Description

Returns the \( \omega \) weights for the detected outliers.

Returns

A double array containing the computed \( \omega \) weights for the detected outliers.

Remarks

If the number of detected outliers equals zero, then an array of length zero is returned. The Compute method must be invoked before using this method. Otherwise, an InvalidOperationException exception is thrown.

GetOutlierFreeForecast

public double[] GetOutlierFreeForecast()

Description

Returns forecasts for the outlier free series.

Returns

A double array of length \( n\text{Forecast} \) containing the forecasts for the outlier free series.

Remarks

Method ComputeForecasts has to be invoked before this method is called. Otherwise, an InvalidOperationException exception is thrown.

GetOutlierFreeSeries

public double[] GetOutlierFreeSeries()

Description

Returns the outlier free series.

Returns

A double array containing the outlier free series.

Remarks

The Compute method must be called before invoking this method. Otherwise, an InvalidOperationException exception is thrown.

GetOutlierStatistics

public int[,] GetOutlierStatistics()
Description
Returns the outlier statistics.

Returns
An int matrix of length nOutliers by 2, where nOutliers is the number of detected outliers, containing the outlier statistics. The first column contains the time at which the outlier was observed (time ranging from 1 to z.Length, the number of observations in the time series) and the second column contains an identifier indicating the type of outlier observed. Outlier types fall into one of five categories:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Outlier Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INNOVATIONAL=0</td>
<td>Innovational Outliers (IO)</td>
</tr>
<tr>
<td>ADDITIVE=1</td>
<td>Additive Outliers (AO)</td>
</tr>
<tr>
<td>LEVEL_SHIFT=2</td>
<td>Level Shift Outliers (LS)</td>
</tr>
<tr>
<td>TEMPORARY_CHANGE=3</td>
<td>Temporary Change Outliers (TC)</td>
</tr>
<tr>
<td>UNABLE_TO_IDENTIFY=4</td>
<td>Unable to Identify (UI)</td>
</tr>
</tbody>
</table>

Remarks
If the number of detected outliers equals zero, then an int array of size 0 is returned. The Compute method must be invoked first before invoking this method. Otherwise, an InvalidOperationException exception is thrown.

GetPsiWeights
public double[] GetPsiWeights()

Description
Returns the $\psi$ weights of the infinite order moving average form of the model.

Returns
A double array of length nForecast containing the $\psi$ weights of the infinite order moving average form of the model for the outlier free series.

Remarks
Method ComputeForecasts must be invoked before this method is called. Otherwise, an InvalidOperationException exception is thrown.

GetResidual
public double[] GetResidual()

Description
Returns the residuals.

Returns
A double array containing the residuals for the outlier free series at the final parameter estimation point.
Remarks
The Compute method must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.

GetTauStatistics
public double[] GetTauStatistics()

Description
Returns the t value for each detected outlier.

Returns
A double array containing the t statistics for each detected outlier.

Remarks
If the number of detected outliers equals zero, then a vector of length 0 is returned. The Compute method must be invoked before using this method. Otherwise, an InvalidOperationException exception is thrown.

Example 1: Parameter estimation and outlier detection for a time series from biology
This example is based on estimates of the Canadian lynx population. Class ARMAOutlierIdentification is used to fit an ARIMA(2,2,0) model of the form

$$(1-B)(1-\phi_1 B - \phi_2 B^2)Y_t = a_t, t = 1,2,\ldots,144,$$

$\{a_t\}$ Gaussian White noise, to the given series. Method Compute determines parameters $\phi_1 = 0.10682$ and $\phi_2 = -0.19666$ and identifies a LS outlier at time point $t = 16$.

using System;
using Imsl.Stat;

public class ARMAOutlierIdentificationEx1 {

    public static void Main(String[] args) {
        double[] series = {
            0.24300e01, 0.25060e01, 0.27670e01, 0.29400e01, 0.31690e01, 0.34500e01,
            0.35940e01, 0.37740e01, 0.36950e01, 0.34110e01, 0.27180e01, 0.19910e01,
            0.22650e01, 0.24460e01, 0.26120e01, 0.33590e01, 0.34110e01, 0.31690e01,
            0.34500e01, 0.35940e01, 0.37740e01, 0.36950e01, 0.34110e01, 0.27180e01,
            0.22650e01, 0.24460e01, 0.26120e01, 0.33590e01, 0.34110e01, 0.31690e01,
            0.34500e01, 0.35940e01, 0.37740e01, 0.36950e01, 0.34110e01, 0.27180e01,
            0.22650e01, 0.24460e01, 0.26120e01, 0.33590e01, 0.34110e01, 0.31690e01,
            0.34500e01, 0.35940e01, 0.37740e01, 0.36950e01, 0.34110e01, 0.27180e01,
            0.22650e01, 0.24460e01, 0.26120e01, 0.33590e01, 0.34110e01, 0.31690e01,
            0.34500e01, 0.35940e01, 0.37740e01, 0.36950e01, 0.34110e01, 0.27180e01,
            0.22650e01, 0.24460e01, 0.26120e01, 0.33590e01, 0.34110e01, 0.31690e01,
            0.34500e01, 0.35940e01, 0.37740e01, 0.36950e01, 0.34110e01, 0.27180e01,
            0.22650e01, 0.24460e01, 0.26120e01, 0.33590e01, 0.34110e01, 0.31690e01,
            0.34500e01, 0.35940e01, 0.37740e01, 0.36950e01, 0.34110e01, 0.27180e01,
            0.22650e01, 0.24460e01, 0.26120e01, 0.33590e01, 0.34110e01, 0.31690e01,
            0.34500e01, 0.35940e01, 0.37740e01, 0.36950e01, 0.34110e01, 0.27180e01,
        };

        //...
int[] model = { 2, 0, 1, 2 };  double[] outlierFreeSeries;  double resStdErr, aic, constant;  double[] ar;  int[,] outlierStatistics;  int numOutliers;

ARMAOutlierIdentification armaOutlier = new ARMAOutlierIdentification(series);

armaOutlier.CriticalValue = 3.5;
armaOutlier.Compute(model);

outlierFreeSeries = armaOutlier.GetOutlierFreeSeries();
numOutliers = armaOutlier.NumberOfOutliers;
constant = armaOutlier.Constant;
ar = armaOutlier.GetAR();
resStdErr = armaOutlier.ResidualStandardError;
aic = armaOutlier.AIC;

Console.Out.WriteLine();
Console.Out.WriteLine();
Console.Out.WriteLine(" ARMA parameters:");
Console.Out.WriteLine("constant:{0,9:f6}", constant);
Console.Out.WriteLine("ar[0]:{0,12:f6}", ar[0]);
Console.Out.WriteLine("ar[1]:{0,12:f6}", ar[1]);

Console.Out.WriteLine("Number of outliers:{0,3:d}", numOutliers);
Console.Out.WriteLine("\n Outlier statistics:");
for (int i = 0; i < numOutliers; i++)
    Console.Out.WriteLine("{0,10:d}{1,18:d}", outlierStatistics[i,0],
    outlierStatistics[i,1]);
Console.Out.WriteLine("\nRSE:{0,11:f6}", resStdErr);
Console.Out.WriteLine("AIC:{0,11:f6}", aic);

for (int i = 0; i < 36; i++)
    Console.Out.WriteLine("{0,4:d}{1,11:f4}{2,15:f4}", i + 1,
    series[i], outlierFreeSeries[i]);

Output

ARMA parameters:
constant: 0.000000

932 ● ARMAOutlierIdentification IMSL C# Numerical Library
ar[0]: 0.106532
ar[1]: -0.195856

Number of outliers: 1

Outlier statistics:
<table>
<thead>
<tr>
<th>Time point</th>
<th>Outlier type</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2</td>
</tr>
</tbody>
</table>

RSE: 0.319542
AIC: 282.918191

Extract from the series:

<table>
<thead>
<tr>
<th>time</th>
<th>original</th>
<th>outlier free</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.4300</td>
<td>2.4300</td>
</tr>
<tr>
<td>2</td>
<td>2.5060</td>
<td>2.5060</td>
</tr>
<tr>
<td>3</td>
<td>2.7670</td>
<td>2.7670</td>
</tr>
<tr>
<td>4</td>
<td>2.9400</td>
<td>2.9400</td>
</tr>
<tr>
<td>5</td>
<td>3.1690</td>
<td>3.1690</td>
</tr>
<tr>
<td>6</td>
<td>3.4500</td>
<td>3.4500</td>
</tr>
<tr>
<td>7</td>
<td>3.5940</td>
<td>3.5940</td>
</tr>
<tr>
<td>8</td>
<td>3.7740</td>
<td>3.7740</td>
</tr>
<tr>
<td>9</td>
<td>3.6950</td>
<td>3.6950</td>
</tr>
<tr>
<td>10</td>
<td>3.4110</td>
<td>3.4110</td>
</tr>
<tr>
<td>11</td>
<td>2.7180</td>
<td>2.7180</td>
</tr>
<tr>
<td>12</td>
<td>1.9910</td>
<td>1.9910</td>
</tr>
<tr>
<td>13</td>
<td>2.2650</td>
<td>2.2650</td>
</tr>
<tr>
<td>14</td>
<td>2.4460</td>
<td>2.4460</td>
</tr>
<tr>
<td>15</td>
<td>2.6120</td>
<td>2.6120</td>
</tr>
<tr>
<td>16</td>
<td>3.3590</td>
<td>2.6997</td>
</tr>
<tr>
<td>17</td>
<td>3.4290</td>
<td>2.7697</td>
</tr>
<tr>
<td>18</td>
<td>3.5330</td>
<td>2.8737</td>
</tr>
<tr>
<td>19</td>
<td>3.2610</td>
<td>2.6017</td>
</tr>
<tr>
<td>20</td>
<td>2.6120</td>
<td>1.9527</td>
</tr>
<tr>
<td>21</td>
<td>2.1790</td>
<td>1.5197</td>
</tr>
<tr>
<td>22</td>
<td>1.6530</td>
<td>0.9937</td>
</tr>
<tr>
<td>23</td>
<td>1.8320</td>
<td>1.1727</td>
</tr>
<tr>
<td>24</td>
<td>2.3280</td>
<td>1.6687</td>
</tr>
<tr>
<td>25</td>
<td>2.7370</td>
<td>2.0777</td>
</tr>
<tr>
<td>26</td>
<td>3.0140</td>
<td>2.3547</td>
</tr>
<tr>
<td>27</td>
<td>3.3280</td>
<td>2.6687</td>
</tr>
<tr>
<td>28</td>
<td>3.4040</td>
<td>2.7447</td>
</tr>
<tr>
<td>29</td>
<td>2.9810</td>
<td>2.3217</td>
</tr>
<tr>
<td>30</td>
<td>2.5570</td>
<td>1.8977</td>
</tr>
<tr>
<td>31</td>
<td>2.5760</td>
<td>1.9167</td>
</tr>
<tr>
<td>32</td>
<td>2.3520</td>
<td>1.6927</td>
</tr>
<tr>
<td>33</td>
<td>2.5560</td>
<td>1.8967</td>
</tr>
<tr>
<td>34</td>
<td>2.8640</td>
<td>2.2047</td>
</tr>
<tr>
<td>35</td>
<td>3.2140</td>
<td>2.5547</td>
</tr>
<tr>
<td>36</td>
<td>3.4350</td>
<td>2.7757</td>
</tr>
</tbody>
</table>
Example 2: Parameter estimation and outlier detection for an artificial time series

This example is an artificial realization of an ARMA(1, 1) process via formula

\[ Y_t - 0.8Y_{t-1} = 10.0 + a_t + 0.5a_{t-1}, \quad t = 1, \ldots, 300. \]

\( \{a_t\} \) Gaussian white noise, \( E[Y_t] = 50.0 \). An additive outlier with \( \omega_1 = 4.5 \) was added at time point \( t = 150 \), a temporary change outlier with \( \omega_2 = 3.0 \) was added at time point \( t = 200 \).

using System;
using Imsl.Stat;

public class ARMAOutlierIdentificationEx2
{
    public static void Main(String[] args)
    {
        double resStdErr, aic;
        int[,] outlierStatistics;
        int numOutliers;
        double[] omegaWeights;
        int[] model = { 1, 1, 1, 0 };
        double[] outlierFreeSeries;
        double constant;
        double[] ar, ma;

        double[] series = { 50.0000000, 50.2728081, 50.6242599, 51.0373917, 51.9317627, 50.3494759, 51.6597252, 52.7004929, 53.5499802, 53.1673279, 50.2373505, 49.3373871, 49.5516472, 48.6692696, 47.6606636, 46.8774185, 45.7315454, 45.6469727, 45.9882356, 45.5216560, 46.0479660, 48.1958656, 48.6387749, 49.9055367, 49.8077278, 47.7858467, 47.9366749, 49.7691956, 48.5425873, 49.1239853, 49.8518791, 50.3320694, 50.9146347, 51.8772049, 51.8745689, 52.3394470, 52.7273712, 51.4310036, 50.6727448, 50.8370399, 51.2843437, 51.8629128, 51.6933670, 49.7038231, 49.0189247, 49.4557030, 50.2718010, 49.9605980, 51.3775749, 50.2285385, 48.2692299, 47.6495590, 49.2938499, 49.1924858, 49.6449242, 50.0446815, 51.9972496, 54.2576981, 52.9835434, 50.4193535, 50.3617897, 51.8276901, 53.1239929, 54.0682144, 54.9238319, 55.6877632, 54.8896332, 54.0701065, 52.2764097, 52.2523254, 53.1248703, 51.1287193, 50.5007815, 49.6504173, 47.2453079, 45.4555626, 45.8449707, 45.9765129, 45.7682228, 45.2343674, 46.6496811, 47.0894432, 49.3368340, 50.8058052, 49.9132500, 48.5932828, 48.2470627, 46.9779968, 45.6760864, 45.7070389, 46.6184909, 47.5303612, 47.5630417, 47.0389214, 46.0352287, 45.8161545, 45.7974396, 46.0015373, 45.3796465, 45.3461685, 47.644016, 49.3327446, 49.3810692, 50.2027817, 51.4567032, 52.3986320, 52.5819206, 52.7721825, 52.6919098, 53.3274345, 55.1345940, 56.8982631, 55.7791634, 55.0616989, 52.3551178, 51.3264084, 51.0968323, 51.1980476, 52.8001442, 52.0545082, 50.8742943, 51.5150337, 51.2242050, 50.5033989, 48.7760124, 47.4179192, 49.7319527, 51.3320541, 52.3918304, 52.4104034, 51.0845947, 49.6485748, 50.6893463, 52.9840813, 53.3246994, 52.4568024, 51.9160911, 53.6683121, 53.455539, 51.7755814, 49.2915611, 49.8755112, 49.4564776, 48.6179193, 49.9643021, 49.3766441, 49.2551308, 50.1021881, 51.0769119, 55.8328133, 52.0212708, 53.4930808, 53.2147255, 52.2356453, 51.9688819, 52.1816330, 51.9880701, 52.5623627, 51.0717278, 52.2431946, 53.6943054, 54.3752098, 54.1492615, 53.8523254, 52.1093712, 52.3982697, 51.2405128, 50.3018112,
51.3819618, 49.5479546, 47.5024452, 47.4447708, 47.8939056, 48.4070015,  48.2440681, 48.7389755, 49.1998024, 49.5798340, 51.1196213,  50.628414, 50.3971405, 51.6084099, 52.4564743, 51.6443901, 52.4080658,  52.4643564, 52.6257210, 53.1604691, 51.930731, 51.4137230, 52.123368,  52.9867249, 53.3180733, 51.9647636, 50.7947655, 52.3815842, 50.8353729,  49.4136009, 52.8355217, 52.2234840, 51.1392517, 48.5245132, 46.8700218,  46.1607285, 45.2324257, 47.4157829, 48.9989090, 49.6230736, 50.4352913,  51.1652985, 50.2586654, 50.7820129, 51.0448799, 51.2880516, 49.689804,  49.0288200, 49.9338837, 48.2214432, 46.2103348, 46.9550171, 47.5595894,  47.7176018, 48.4502945, 50.9816895, 51.6950073, 51.6973495, 52.1941261,  51.8988075, 52.5617599, 52.0218391, 49.5236053, 47.9684906, 48.2445183,  48.8275146, 49.7176971, 51.5649338, 52.5627213, 52.0182419, 50.968835,  51.5846901, 50.9486771, 48.8685837, 48.5600624, 48.4760994, 48.5348396,  50.4187813, 51.2542831, 50.1872864, 50.4407692, 50.6222687, 50.4972000,  51.0036087, 51.3367500, 51.7362020, 53.0463791, 53.6261253, 52.0728683,  48.9740753, 49.3208730, 49.2733917, 49.8519020, 50.8562126, 49.5594254,  49.6109200, 48.3785629, 48.0026474, 49.4874268, 50.1596375, 51.8059540,  53.0288620, 51.3521075, 49.3118415, 48.7993906, 47.7201881, 46.343914,  46.5303612, 47.6299632, 48.6012459, 47.8586757, 48.0604057, 47.1352806,  49.5724792, 50.5566483, 49.4182968, 50.5578079, 50.6837376, 50.6333389,  51.9766159, 51.0595245, 49.3761640, 49.9667702, 47.1658173, 47.4411278,  47.5360374, 48.9914742, 50.4774620, 50.2728043, 51.9117165, 53.7627792;

ARMAOutlierIdentification armaOutlier = new ARMAOutlierIdentification(series);
armaOutlier.RelativeError = 1.0e-5;
armaOutlier.Compute(model);

outlierFreeSeries = armaOutlier.GetOutlierFreeSeries();
umOutliers = armaOutlier.NumberOfOutliers;
omegaWeights = armaOutlier.GetOmegaWeights();
constant = armaOutlier.Constant;
ar = armaOutlier.GetAR();
ma = armaOutlier.GetMA();
resStdErr = armaOutlier.ResidualStandardError;
aic = armaOutlier.AIC;

Console.Out.WriteLine("\n ARMA parameters:");
Console.Out.WriteLine("constant:{0, 11:f6}", constant);
Console.Out.WriteLine("ar[0]:{0,14:f6}", ar[0]);
Console.Out.WriteLine("ma[0]:{0,14:f6}", ma[0]);

for (int i = 0; i < numOutliers; i++)
{
    Console.Out.WriteLine("{0,10:d}{1,18:d}", outlierStatistics[i,0],
        outlierStatistics[i,1]);
}

for (int i = 0; i < numOutliers; i++)
{
    Console.Out.WriteLine("{0,10:d}{1,11:f6}", omegaWeights[i]);
}
Console.Out.WriteLine("\nRSE:{0,12:f6}", resStdErr);
Output

ARMA parameters:
constant: 10.829171
ar[0]: 0.785217
ma[0]: -0.496449

Number of outliers: 2

Outlier statistics:
Time point Outlier type
150 1
200 3

Omega statistics:
Time point Omega
150 4.477853
200 3.381622

RSE: 1.007223
AIC: 1417.044262

Example 3: Forecasting an outlier contaminated time series

This example is a realization of an ARMA(2,1) process described by the model

\[ Y_t - Y_{t-1} + 0.24 Y_{t-2} = 10.0 + a_t + 0.5 a_{t-1}, \]

\( \{a_t\} \) a Gaussian White noise process. An additive outlier with \( \omega_1 = 4.5 \) was added at time point \( t = 150 \), a temporary change outlier with \( \omega_2 = 3.0 \) was added at time point \( t = 200 \).

Outliers were artificially added to the outlier free series \( \{Y_t\}_{t=1,\ldots,280} \) at time points \( t = 150 \) (level shift with \( \omega_1 = +2.5 \)) and \( t = 200 \) (additive outlier with \( \omega_2 = +3.2 \)), resulting in the outlier contaminated series \( \{Z_t\}_{t=1,\ldots,280} \). For both series, forecasts were determined for time points \( t = 281, \ldots, 290 \) and compared with the actual values of the series.
double constant;
double[] ar, ma;
int nForecast = 10;
double[] outlierContaminatedForecast;
double[] outlierFreeForecast;
double[] probabilityLimits;
double[] psiWeights;
double[] outlierContaminatedSeries = {
41.6699982, 41.6699982, 42.072144, 42.6123962, 43.6161919, 42.1932831, 
43.105450, 44.3518715, 38.6873589, 37.9272423, 36.8310852, 
37.4254879, 37.3440933, 37.961374, 40.3810501, 41.3646622, 42.6495285, 
42.6096764, 40.3134537, 39.7971268, 41.5401535, 40.7160759, 41.0363541, 
41.8171833, 42.4190292, 43.0318832, 43.9968109, 44.0419977, 44.3225212, 
44.6082611, 43.2199961, 42.0419977, 41.9679718, 42.4926224, 43.2091255, 
43.2512283, 42.2367174, 40.1057358, 40.4510574, 41.5329170, 44.0419977, 
43.009141, 42.1592140, 41.9679718, 42.4926224, 43.2091255, 43.2512283, 
43.2912283, 42.9420026, 43.771248, 43.6518707, 37.0550499, 46.947708, 
44.1835632, 40.3352280, 38.5232544, 39.4119797, 41.8316803, 43.7091446, 
46.700745, 45.4120026, 42.7635345, 43.7112923, 42.0768661, 43.7112923, 
43.7112923, 42.1592140, 41.9679718, 42.4926224, 43.2091255, 43.2512283, 
43.2512283, 42.8811569, 44.4796944, 44.6887932, 43.1670265, 42.226143, 
43.1670265, 42.226143, 41.830154, 44.3719249, 44.4174194, 43.5068550, 44.973015, 
44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 44.973015, 43.5068550, 
43.5068550};
// Actual values of the outlier contaminated series for
t = 181,...,190
double[] exactForecastOutlierContaminatedSeries = {
  42.6384087, 41.7088661, 43.9399033, 45.4284401, 44.4558411,
  45.1761856, 45.3489113, 45.1892662, 46.3754730, 45.6082802};

// Actual values of the outlier free series for
// t = 181,...,190. This are the values of the outlier contaminated
// series - 2.5
double[] exactForecastOutlierFreeSeries = {
  40.1384087, 39.2088661, 41.4399033, 42.9284401, 41.9558411,
  42.6761856, 42.8489113, 42.6892662, 43.8754730, 43.1082802};

ARMAOutlierIdentification armaOutlier =
    new ARMAOutlierIdentification(outlierContaminatedSeries);
armaOutlier.RelativeError = 1.0e-5;
armaOutlier.Compute(model);
armaOutlier.ComputeForecasts(nForecast);
numOutliers = armaOutlier.NumberOfOutliers;
constant = armaOutlier.Constant;
ar = armaOutlier.GetAR();
ma = armaOutlier.GetMA();
resStdErr = armaOutlier.ResidualStandardError;
aic = armaOutlier.AIC;
outlierContaminatedForecast = armaOutlier.GetForecast();
outlierFreeForecast = armaOutlier.GetOutlierFreeForecast();
probabilityLimits = armaOutlier.GetDeviations();
psiWeights = armaOutlier.GetPsiWeights();
Console.Out.WriteLine("\n\n ARMA parameters:");
Console.Out.WriteLine("constant: {0,9:f6}", constant);
Console.Out.WriteLine("ar[0]: {0,12:f6}", ar[0]);
Console.Out.WriteLine("ar[1]: {0,12:f6}", ar[1]);
Console.Out.WriteLine("ma[0]: {0,12:f6}", ma[0]);
Console.Out.WriteLine();
Console.Out.WriteLine("Number of outliers: {0,4:d}", numOutliers);
Console.Out.WriteLine();
Console.Out.WriteLine(" Outlier statistics:"); 
Console.Out.WriteLine("Time point  Outlier type");
for (int i = 0; i < numOutliers; i++)
    Console.Out.WriteLine("{0,10:d}{1,20:d}", outlierStatistics[i,0],
        outlierStatistics[i,1]);
Console.Out.WriteLine("\nRSE:{0,12:f6}", resStdErr);
Console.Out.WriteLine("AIC:{0,12:f6}", aic);
Console.Out.WriteLine();
Console.Out.WriteLine(" * * * Forecast Table for outlier "
    + "contaminated series * * *");
Console.Out.WriteLine();
Console.Out.WriteLine(" Exact forecast"+
    " limit psi");
for (int i = 0; i < nForecast; i++)
  Console.Out.WriteLine("\n\n ARMA Outlier Identification IMSL C# Numerical Library")
Console.Out.WriteLine(" {0,8:f4}{1,13:f4}{2,12:f4}{3,12:f4}",
    exactForecastOutlierContaminatedSeries[i],
    outlierContaminatedForecast[i],
    probabilityLimits[i],
    psiWeights[i]);

Console.Out.WriteLine("\n\n   * * * Forecast Table for outlier " +
    "free series * * *");
Console.Out.WriteLine();
Console.Out.WriteLine("   Exact forecast " +
    "limit  psi");
for (int i = 0; i < nForecast; i++)
    Console.Out.WriteLine("   {0,8:f4}{1,13:f4}{2,12:f4}{3,12:f4}",
        exactForecastOutlierFreeSeries[i],
        outlierFreeForecast[i],
        probabilityLimits[i],
        psiWeights[i]);
}
}

Output

ARMA parameters:
constant:  8.891421
ar[0]:    0.944052
ar[1]:    -0.150404
ma[0]:    -0.558928

Number of outliers:  2

Outlier statistics:
Time point   Outlier type
   150       2
   200       1

RSE:   1.004306
AIC:  1323.617443

* * * Forecast Table for outlier contaminated series * * *

  Exact forecast limit  psi
42.6384  42.3158  1.9684  1.5030
41.7089  42.7933  3.5535  1.2685
43.9399  43.2822  4.3430  0.9715
45.4284  43.6718  4.7453  0.7263
44.4558  43.9662  4.9560  0.5396
45.1762  44.1854  5.0686  0.4002
45.3489  44.3482  5.1294  0.2966
45.1893  44.4688  5.1625  0.2198
46.3755  44.5582  5.1806  0.1629
45.6083  44.6245  5.1906  0.1207

* * * Forecast Table for outlier free series * * *

Time Series and Forecasting   ARMAOutlierIdentification   939
<table>
<thead>
<tr>
<th>Exact forecast limit psi</th>
</tr>
</thead>
<tbody>
<tr>
<td>40.1384 40.5904 1.9684 1.5030</td>
</tr>
<tr>
<td>39.2089 41.0679 3.5535 1.2685</td>
</tr>
<tr>
<td>41.4399 41.5567 4.3430 0.9715</td>
</tr>
<tr>
<td>42.9284 41.9464 4.7453 0.7263</td>
</tr>
<tr>
<td>41.9558 42.2407 4.9560 0.5396</td>
</tr>
<tr>
<td>42.6762 42.4600 5.0686 0.4002</td>
</tr>
<tr>
<td>42.8489 42.6227 5.1294 0.2966</td>
</tr>
<tr>
<td>42.6893 42.7434 5.1625 0.2198</td>
</tr>
<tr>
<td>43.8755 42.8328 5.1806 0.1629</td>
</tr>
<tr>
<td>43.1083 42.8991 5.1906 0.1207</td>
</tr>
</tbody>
</table>

**AutoARIMA Class**

```csharp
public class Imsl.Stat.AutoARIMA

Automatically identifies time series outliers, determines parameters of a multiplicative seasonal ARIMA \((p, 0, q) \times (0, d, 0)\) model and produces forecasts that incorporate the effects of outliers whose effects persist beyond the end of the series.

Class `AutoARIMA` determines the parameters of a multiplicative seasonal ARIMA \((p, 0, q) \times (0, d, 0)\) model, and then uses the fitted model to identify outliers and prepare forecasts. The order of this model can be specified or automatically determined through use of an overloaded `Compute` method. Potential missing values in the time series are estimated prior to the parameter and outlier computations.

The ARIMA \((p, 0, q) \times (0, d, 0)\) model handled by class `AutoARIMA` has the following form:

\[
\phi(B)\Delta^d(Y_t - \mu) = \theta(B)a_t, \quad t = 1, 2, \ldots, n,
\]

where

\[
\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p, \quad \theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \ldots - \theta_q B^q, \quad \Delta^d = (1 - B^s)^d
\]

and

\[
B^k Y_t = Y_{t-k}
\]

It is assumed that all roots of \(\phi(B)\) and \(\theta(B)\) lie outside the unit circle. Clearly, if \(s = 1\) the model reduces to the traditional ARIMA \((p, d, q)\) model.

\(Y_t\) is the unobserved, outlier-free time series with mean \(\mu\), and white noise \(a_t\). This model is referred to as the underlying, outlier-free model. Class `AutoARIMA` does not assume that this series is observable. It assumes that the observed values might be contaminated by one or more outliers, whose effects are added to the underlying outlier-free series:

\[
Y_t^* = Y_t + \text{outlierEffect}_t
\]
Outlier identification uses the algorithm developed by Chen and Liu (1993). Outliers are classified into 1 of 5 types:

1. innovational
2. additive
3. level shift
4. temporary change and
5. unable to identify

Once the model parameters are estimated and the outliers are identified, class AutoARIMA estimates $Y_t$, the outlier-free series representation of the data, by removing the estimated outlier effects. Parameter estimation and outlier detection are based on methods from class ARMAOutlierIdentification.

Using the information about the adjusted ARIMA $(p,0,q) \times (0,d,0)$, model and the removed outliers, forecasts are then prepared for the outlier-free series. Outlier effects are added to these forecasts to produce a forecast for the observed series, $Y'_t$. If there are no outliers, then the forecasts for the outlier-free series and the observed series will be identical.

**Model selection techniques**

Users have an option of either specifying specific values for $p$, $q$, $s$, $d$ or have class AutoARIMA automatically select best fit values. Model selection can be conducted in one of three ways listed below depending upon which Compute method of class AutoARIMA is invoked.

**Technique 1: Automatic ARIMA $(p,0,0) \times (0,d,0)_s$ Selection**

This technique, chosen by use of method Compute(int maxlag), tries to fit a model of the form

$$\phi(B)A^d_s(Y_t - \mu) = a_t$$

to the outlier free series $Y_t$.

It initially searches for the AR($p$) representation with minimum value of the chosen information criterion (AIC, AICC or BIC) for the noisy data, where $p = 0, \ldots, \text{maxlag}$.

If the user calls methods SetPeriods and SetDifferenceOrders prior to invoking the Compute method, then the values in arrays periods and orders are included in the search to find an optimum ARIMA $(p,0,0) \times (0,d,0)_s$ representation of the series. Here, every possible combination of values for $s, d$ in periods and orders, respectively, are examined. The best found model order is then used as input for the parameter and outlier detection routine.

The optimum values for $p$, $q$, $s$ and $d$ are returned through method GetOptimumModelOrder.

**Technique 2: Grid Search**

This technique, chosen by means of method Compute(int[] arOrders, int[] maOrders), conducts a grid search for $p$ and $q$ using all possible combinations of candidate values in arOrders and maOrders.
If methods `SetPeriods` and `SetDifferenceOrders` are called prior to invoking the `Compute` method, then the grid search is extended to include the candidate values for `s` and `d` given in arrays `periods` and `orders`, respectively.

If method `SetDifferenceOrders` is not called prior to `Compute`, then `d = 0` by default, and therefore no seasonal adjustment is attempted. The grid search is then restricted to searching for optimum values of `p` and `q` only.

The optimum values for `p`, `q`, `s` and `d` are contained in the array returned by method `GetOptimumModelOrder`.

**Technique 3: Specified ARIMA\((p, 0, q) \times (0, d, 0)\)_s Model**

In the third technique, selectable by means of method `Compute(int p, int q, int s, int d)`, specific values for `p`, `q`, `s` and `d` are given. This technique has essentially the same functionality as class `ARMAOutlierIdentification` but with the additional option of missing value estimation.

**Outliers**

The algorithm of Chen and Liu (1993) is used to identify outliers. The number of outliers identified is returned via the `NumberOfOutliers` property. Both the time and classification for these outliers are contained in the matrix returned by method `GetOutlierStatistics`. Outliers are classified into one of five categories based upon the standardized statistic for each outlier type. The time at which the outlier occurred is given in the first column of the returned matrix. The outlier identifier returned in the second column is according to the descriptions in the following table:
<table>
<thead>
<tr>
<th>Outlier Identifier</th>
<th>Name</th>
<th>General Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INNOVATIONAL = 0</td>
<td>Innovational Outlier (IO)</td>
<td>Innovational outliers persist. That is, there is an initial impact at the time the outlier occurs. This effect continues in a lagged fashion with all future observations. The lag coefficients are determined by the coefficients of the underlying ARIMA($p, 0, q$) × (0, $d, 0$) model.</td>
</tr>
<tr>
<td>ADDITIVE = 1</td>
<td>Additive Outlier (AO)</td>
<td>Additive outliers do not persist. As the name implies, an additive outlier affects only the observation at the time the outlier occurs. Hence additive outliers have no effect on future forecasts.</td>
</tr>
<tr>
<td>LEVEL_SHIFT = 2</td>
<td>Level Shift (LS)</td>
<td>Level shift outliers persist. They have the effect of either raising or lowering the mean of the series starting at the time the outlier occurs. This shift in the mean is abrupt and permanent.</td>
</tr>
<tr>
<td>TEMPORARY_CHANGE = 3</td>
<td>Temporary Change (TC)</td>
<td>Temporary change outliers persist and are similar to level shift outliers with one major exception. Like level shift outliers, there is an abrupt change in the mean of the series at the time this outlier occurs. However, unlike level shift outliers, this shift is not permanent. The TC outlier gradually decays, eventually bringing the mean of the series back to its original value. The rate of this decay is modeled using property $\Delta$. The default of $\Delta = 0.7$ is the value recommended for general use by Chen and Liu (1993).</td>
</tr>
<tr>
<td>UNABLE_TO_IDENTIFY = 4</td>
<td>Unable to Identify (UI)</td>
<td>If an outlier is identified as the last observation, then the algorithm is unable to determine the outlier’s classification. For forecasting, a UI outlier is treated as an IO outlier. That is, its effect is lagged into the forecasts.</td>
</tr>
</tbody>
</table>
Except for additive outliers (AO), the effect of an outlier persists to observations following that outlier. Forecasts produced by methods of class AutoARIMA take this into account.

For more information on forecasting an outlier contaminated series, see the description of class ARMAOutlierIdentification.

### Fields

**ADDITIVE**

```java
public int ADDITIVE
```

**Description**

Indicates detection of an additive outlier.

**INNOVATIONAL**

```java
public int INNOVATIONAL
```

**Description**

Indicates detection of an innovational outlier.

**LEVEL_SHIFT**

```java
public int LEVEL_SHIFT
```

**Description**

Indicates detection of a level shift outlier.

**TEMPORARY_CHANGE**

```java
public int TEMPORARY_CHANGE
```

**Description**

Indicates detection of a temporary change outlier.

**UNABLE_TO_IDENTIFY**

```java
public int UNABLE_TO_IDENTIFY
```

**Description**

Indicates detection of an outlier that cannot be categorized.

### Properties

**AccuracyTolerance**

```java
virtual public double AccuracyTolerance {get; set; }
```
Description
The tolerance value controlling the accuracy of the parameter estimates.

Property Value
A double scalar containing a positive tolerance value controlling the accuracy of parameter estimates during outlier detection.
Default: AccuracyTolerance = 0.001.

AIC
public double AIC {get; }

Description
Akaike's information criterion (AIC) for the optimum model.

Property Value
A double scalar containing Akaike's information criterion (AIC) for the optimum outlier free series.

Remarks
One of the Compute methods must be called before invoking this method. Otherwise, an InvalidOperationException exception is thrown.

AICC
public double AICC {get; }

Description
Akaike's Corrected Information Criterion (AICC) for the optimum model.

Property Value
A double scalar containing Akaike's Corrected Information Criterion (AICC) for the optimum outlier free series.

Remarks
One of the Compute methods must be called before invoking this method. Otherwise, an InvalidOperationException exception is thrown.

BIC
public double BIC {get; }

Description
The Bayesian Information Criterion (BIC) for the optimum model.

Property Value
A double scalar containing the Bayesian Information Criterion (BIC) for the optimum outlier free series.
Remarks
One of the Compute methods must be called before invoking this method. Otherwise, an InvalidOperationException exception is thrown.

**Confidence**
```csharp
public double Confidence {get; set; }
```

**Description**
The confidence level used in the calculation of confidence limit deviations via method GetDeviations.

**Property Value**
A double scalar specifying the confidence level used in computing forecast confidence intervals.
Default: Confidence = 0.95.

Remarks
The confidence levels must be greater than 0.0 and less than 1.0. Typical choices are 0.90, 0.95, and 0.99.

**Constant**
```csharp
public double Constant {get; }
```

**Description**
The constant parameter estimate for the optimum model.

**Property Value**
A double scalar containing the constant parameter estimate for the optimum model.

Remarks
Note that one of the Compute methods must be invoked first before using this property. Otherwise, the method throws an InvalidOperationException exception.

**CriticalValue**
```csharp
public double CriticalValue {get; set; }
```

**Description**
The critical value used as a threshold during outlier detection.

**Property Value**
A double, strictly positive scalar, the critical value used as a threshold for the statistics used in the outlier detection.
Default: CriticalValue = 3.0.

**Delta**
```csharp
public double Delta {get; set; }
```

**Description**
The dampening effect parameter.
Property Value
A double scalar, the dampening effect parameter used in the detection of a Temporary Change Outlier (TC). The scalar must be greater than 0 and less than 1.
Default: Delta = 0.7.

MaximumARLag
public int MaximumARLag {get; set; }

Description
The maximum AR lag used in the determination of the optimum (s,d) combination of method Compute(int[] arOrders, int[] maOrders).

Property Value
A scalar int, the maximum AR lag used in the computation of the optimum (s,d) combination for method Compute(int[] arOrders, int[] maOrders).
Default: MaximumARLag = 10.

Remarks
It is required that the maximum AR lag is greater than zero and smaller than the original series after replacement of potential missing values.

ModelSelectionCriterion
public Imsl.Stat.AutoARIMA.InformationCriterion ModelSelectionCriterion {get; set; }

Description
The model selection criterion used in the optimum model search.

Property Value
An InformationCriterion value specifying the model selection criterion to be used in the search for the optimum model.
Default: InformationCriterion.Akaike is chosen.

Remarks

<table>
<thead>
<tr>
<th>InformationCriterion</th>
<th>Criterion Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akaike</td>
<td>Use Akaike’s information criterion (AIC).</td>
</tr>
<tr>
<td>AkaikeCorrected</td>
<td>Use Akaike’s corrected information criterion (AICC).</td>
</tr>
<tr>
<td>Bayes</td>
<td>Use the Bayesian information criterion (BIC).</td>
</tr>
</tbody>
</table>

NumberOfOutliers
public int NumberOfOutliers {get; }

Description
The number of detected outliers.
Property Value
An int scalar containing the number of outliers detected.

Remarks
One of the Compute methods must be invoked first before using this property. Otherwise, the method throws an InvalidOperationException exception.

RelativeError
public double RelativeError {get; set; }

Description
The stopping criterion for use in the nonlinear equation solver.

Property Value
A double positive scalar containing the stopping criterion for use in the nonlinear equation solver used in the least-squares algorithm.

Default: RelativeError = 2.2204460492503131e-012.

ResidualStandardError
public double ResidualStandardError {get; }

Description
The residual standard error of the outlier free series.

Property Value
A double scalar containing the standard error of the outlier free series.

Remarks
Note that one of the Compute methods must be invoked first before using this property. Otherwise, an InvalidOperationException exception is thrown.

Constructor

AutoARIMA
public AutoARIMA(int[] times, double[] x)

Description
Constructor for AutoARIMA.

Parameters

times – An int array of length nObs, where nObs is the number of observed time series values, containing the time points \( t_1, \ldots, t_{n\text{Obs}} \) at which the time series was observed. It is required that \( t_1, \ldots, t_{n\text{Obs}} \) are in strictly increasing order. Times for missing values are identified as non-incremental gaps in this series. A gap of missing values in \( x \) is assumed if the difference between two consecutive values is greater than 1, i.e. \( t_{i+1} - t_i > 1 \). The difference is the number of missing values in the gap.
x – A double array containing the observations \( Y_1^*, Y_2^*, \ldots, Y_{n_{\text{Obs}}}^* \) at the times given in array \( \text{times} \). This series can contain outliers and missing observations.

## Methods

### Compute

```java
public void Compute(int maxlag)
```

**Description**

Estimates potential missing values, detects and determines outliers and simultaneously fits an optimum model from a set of different ARIMA\((p, 0, 0) \times (0, d, 0)\)_s to the outlier free time series.

**Parameter**

- \( \text{maxlag} \) – The maximum value for \( p \) allowed when fitting ARIMA\((p, 0, 0) \times (0, d, 0)\)_s models to the given series, \( 0 \leq p \leq \text{maxlag} \). It is required that \( 1 \leq \text{maxlag} \leq x.\text{Length} \). The optimum ARIMA\((p, 0, 0) \times (0, d, 0)\)_s model is determined according to the model selection criterion chosen by the user, see property \( \text{ModelSelectionCriterion} \).

**Exceptions**

- `Imsl.Stat.MatrixSingularException` is thrown if the input matrix is singular.
- `Imsl.Stat.TooManyCallsException` is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- `Imsl.Stat.IncreaseErrRelException` is thrown if the bound for the relative error is too small.
- `Imsl.Stat.NewInitialGuessException` is thrown if the iteration has not made good progress.
- `Imsl.Stat.IllConditionedException` is thrown if the problem is ill-conditioned.
- `Imsl.Stat.TooManyIterationsException` is thrown if the maximum number of iterations is exceeded.
- `Imsl.Stat.TooManyFunctionEvaluationsException` is thrown if the maximum number of function evaluations is exceeded.
- `Imsl.Stat.TooManyJacobianEvalException` is thrown if the maximum number of Jacobian evaluations is exceeded.
- `Imsl.Stat.SingularTriangularMatrixException` is thrown if the input matrix to ARAutoUnivariate is singular.
- `Imsl.Stat.NonInvertibleException` is thrown if the intermediate or final maximum likelihood estimates for the time series are noninvertible.
- `Imsl.Stat.NonStationaryException` is thrown if the intermediate or final maximum likelihood estimates for the time series are nonstationary.
- `Imsl.Stat.InitialMAException` is thrown if the initial values provided for the moving average terms are noninvertible. In this case, ARMA\(_{\text{MaxLikelihood}}\) terminates and does not compute the time series estimates.
Imsl.Math.DidNotConvergeException is thrown if the algorithm computing the roots of the AR- or MA- polynomial does not converge.

Imsl.Math.SingularMatrixException is thrown if during the computation of a small perturbation of the matrix product $A^T A$, it is found that $A$, the matrix used in the determination of the $\omega$ weights, is singular.

Imsl.Math.NotSPDException is thrown if during the computation of a small perturbation of the matrix product $A^T A$, it is found that $A$, the matrix used in the determination of the $\omega$ weights, is not positive definite.

Imsl.Stat.NoAcceptableModelFoundException is thrown if no appropriate ARIMA model for the given time series could be found.

---

**Compute**

```csharp
public void Compute(int[] arOrders, int[] maOrders)
```

**Description**

Estimates potential missing values, detects and determines outliers and simultaneously fits an optimum model from a set of different ARIMA $(p, 0, q) \times (0, d, 0)$ models to the outlier free time series.

**Parameters**

- **arOrders** – An int array containing all possible AR orders to consider in the optimum model search. It is required that all values in `arOrders` are greater than or equal to zero.
- **maOrders** – An int array containing all possible MA orders to consider in the optimum model search. It is required that all values in `maOrders` are greater than or equal to zero.

**Exceptions**

- **Imsl.Stat.MatrixSingularException** is thrown if the input matrix is singular.
- **Imsl.Stat.TooManyCallsException** is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- **Imsl.Stat.IncreaseErrRelException** is thrown if the bound for the relative error is too small.
- **Imsl.Stat.NewInitialGuessException** is thrown if the iteration has not made good progress.
- **Imsl.Stat.IllConditionedException** is thrown if the problem is ill-conditioned.
- **Imsl.Stat.TooManyIterationsException** is thrown if the maximum number of iterations is exceeded.
- **Imsl.Stat.TooManyFunctionEvaluationsException** is thrown if the maximum number of function evaluations is exceeded.
- **Imsl.Stat.TooManyJacobianEvalException** is thrown if the maximum number of Jacobian evaluations is exceeded.
- **Imsl.Stat.SingularTriangularMatrixException** is thrown if the input matrix to ARAutoUnivariate is singular.
- **Imsl.Stat.NonInvertibleException** is thrown if the intermediate or final maximum likelihood estimates for the time series are noninvertible.
Imsl.Stat.NonStationaryException is thrown if the intermediate or final maximum likelihood estimates for the time series are nonstationary.

Imsl.Stat.InitialMAException is thrown if the initial values provided for the moving average terms are noninvertible. In this case, ARMAMaxLikelihood terminates and does not compute the time series estimates.

Imsl.Math.DidNotConvergeException is thrown if the algorithm computing the roots of the AR- or MA- polynomial does not converge.

Imsl.Math.SingularMatrixException is thrown if during the computation of a small perturbation of the matrix product $A^T A$, it is found that $A$, the matrix used in the determination of the $\omega$ weights, is singular.

Imsl.Math.NotSPDException is thrown if during the computation of a small perturbation of the matrix product $A^T A$, it is found that $A$, the matrix used in the determination of the $\omega$ weights, is not positive definite.

Imsl.Stat.NoAcceptableModelFoundException is thrown if no appropriate ARIMA model for the given time series could be found.

**Compute**

```java
public void Compute(int p, int q, int s, int d)
```

**Description**

Estimates potential missing values, detects and determines outliers and simultaneously fits an ARIMA($p,0,q$) × (0,$d$,0) model to the outlier free time series.

**Parameters**

- **p** – A non-negative scalar int, the order of the AR part of the model.
- **q** – A non-negative scalar int, the order of the MA part of the model.
- **s** – A positive scalar int, the period of the difference used in the model.
- **d** – A non-negative scalar int, the order of the difference used in the model.

**Exceptions**

- Imsl.Stat.MatrixSingularException is thrown if the input matrix is singular.
- Imsl.Stat_TOO_MANY_CALLS is thrown if the number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
- Imsl.Stat.IncreaseErrRelException is thrown if the bound for the relative error is too small.
- Imsl.Stat.NewInitialGuessException is thrown if the iteration has not made good progress.
- Imsl.Stat.IllConditionedException is thrown if the problem is ill-conditioned.
- Imsl.Stat_TOO_MANY_ITERATIONS is thrown if the maximum number of iterations is exceeded.
- Imsl.Stat_TOO_MANY_FUNCTION_EVALUATIONS is thrown if the maximum number of function evaluations is exceeded.

---

**Time Series and Forecasting**

AutoARIMA • 951
Imsl.Stat.TooManyJacobianEvalException is thrown if the maximum number of Jacobian evaluations is exceeded.

Imsl.Stat.SingularTriangularMatrixException is thrown if the input matrix to ARAutoUnivariate is singular.

Imsl.Stat.NonInvertibleException is thrown if the intermediate or final maximum likelihood estimates for the time series are noninvertible.

Imsl.Stat.NonStationaryException is thrown if the intermediate or final maximum likelihood estimates for the time series are nonstationary.

Imsl.Stat.InitialMAException is thrown if the initial values provided for the moving average terms are noninvertible. In this case, ARMAMaxLikelihood terminates and does not compute the time series estimates.

Imsl.Math.DidNotConvergeException is thrown if the algorithm computing the roots of the AR- or MA- polynomial does not converge.

Imsl.Math.SingularMatrixException is thrown if during the computation of a small perturbation of the matrix product \( A^T A \), it is found that \( A \), the matrix used in the determination of the \( \omega \) weights, is singular.

Imsl.Math.NotSPDException is thrown if during the computation of a small perturbation of the matrix product \( A^T A \), it is found that \( A \), the matrix used in the determination of the \( \omega \) weights, is not positive definite.

Imsl.Stat.NoAcceptableModelFoundException is thrown if no appropriate ARIMA model for the given time series could be found.

**Forecast**

```csharp
public void Forecast(int nForecast)
```

**Description**

Computes forecasts, associated probability limits and \( \psi \) weights for the given outlier contaminated time series.

**Parameter**

- `nForecast` – An int scalar representing the number of forecasts that will be computed.

- `nForecast` must be greater than 0. Forecast origin is the time point of the last observed value in the time series, \( t_{nObs} \). Forecasts are computed for lead times 1, 2, ..., \( nForecast \), i.e. time points \( t_{nObs} + 1, t_{nObs} + 2, ..., t_{nObs} + nForecast \).

**Remarks**

One of the Compute methods must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.

**GetAR**

```csharp
public double[] GetAR()
```

**Description**

Returns the final autoregressive parameter estimates of the optimum model.
Returns
A double array containing the final autoregressive parameter estimates. If the optimum model has no AR component then a zero-length array is returned.

Remarks
Note that one of the Compute methods must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.

GetCompleteTimes
public int[] GetCompleteTimes()

Description
Returns all time points at which the original series was observed, including values for times with missing values in x.

Returns
An int array of length times.Length-1-times[0]+1 containing the times at which the time series (including missing values) was observed.

Remarks
One of the Compute methods must be called before invoking this method. Otherwise, an InvalidOperationException exception is thrown.

GetCompleteTimeSeries
public double[] GetCompleteTimeSeries()

Description
Returns the original series with potentially missing values replaced by estimates.

Returns
A double array containing the original time series with missing values replaced by estimates.

Remarks
One of the Compute methods must be called before invoking this method. Otherwise, an InvalidOperationException exception is thrown.

GetDeviations
public double[] GetDeviations()

Description
Returns the deviations used for calculating the forecast confidence limits.

Returns
A double array of length nForecast containing the deviations from each forecast for calculating forecast confidence intervals. The confidence level is specified by the Confidence property.
Remarks
Method Forecast must be invoked before this method is called. Otherwise, an InvalidOperationException exception is thrown.

GetForecast
public double[] GetForecast()

Description
Returns forecasts for the original outlier contaminated series.

Returns
A double array of length nForecast containing the forecasts for the original series. Forecast origin is the time point of the last observed value in the time series, t_nObs. Forecasts are returned for lead times 1, 2, ..., nForecast, i.e. time points t_{nObs} + 1, t_{nObs} + 2, ..., t_{nObs} + nForecast.

Remarks
Method Forecast must be invoked before this method is called. Otherwise, an InvalidOperationException exception is thrown.

GetMA
public double[] GetMA()

Description
Returns the final moving average parameter estimates of the optimum model.

Returns
A double array containing the final moving average parameter estimates. If the optimum model has no MA component then a zero-length array is returned.

Remarks
Note that one of the Compute methods must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.

GetOptimumModelOrder
public int[] GetOptimumModelOrder()

Description
Returns the order (p, 0, q) × (0, d, 0), of the optimum model.

Returns
An int array of length 4 containing the values p, q, s and d for the optimum model.

Remarks
One of the Compute methods must be invoked first before calling this method. Otherwise, an InvalidOperationException exception is thrown.

GetOutlierFreeForecast
public double[] GetOutlierFreeForecast()
Description
Returns forecasts for the outlier free series.

Returns
A double array of length nForecast containing the forecasts for the outlier free series. Forecast origin is the time point of the last observed value in the time series, t_{nObs}. Forecasts are returned for lead times 1, 2, ..., nForecast, i.e. time points t_{nObs} + 1, t_{nObs} + 2, ..., t_{nObs} + nForecast.

Remarks
Method Forecast must be invoked before this method is called. Otherwise, an InvalidOperationException exception is thrown.

GetOutlierFreeSeries
public double[] GetOutlierFreeSeries()

Description
Returns the outlier free series.

Returns
A double array containing the original time series with estimated missing values after removal of any outlier effects.

Remarks
One of the Compute methods must be called before invoking this method. Otherwise, an InvalidOperationException exception is thrown.

GetOutlierStatistics
public int[,] GetOutlierStatistics()

Description
Returns the outlier statistics.

Returns
A double array of length nOutliers by 2, where nOutliers is the number of detected outliers, containing the outlier statistics. The first column contains the time at which the outlier was observed (time ranging from times[0] to times[times.Length - 1]) and the second column contains an identifier indicating the type of outlier observed.

Remarks
Outlier types fall into one of five categories:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Outlier type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INNOVATIONAL = 0</td>
<td>Innovational Outliers (IO)</td>
</tr>
<tr>
<td>ADDITIVE = 1</td>
<td>Additive Outliers (AO)</td>
</tr>
<tr>
<td>LEVEL_SHIFT = 2</td>
<td>Level Shift Outliers (LS)</td>
</tr>
<tr>
<td>TEMPORARY_CHANGE = 3</td>
<td>Temporary Change Outliers (TC)</td>
</tr>
<tr>
<td>UNABLE_TO_IDENTIFY = 4</td>
<td>Unable to Identify (UI)</td>
</tr>
</tbody>
</table>
If the number of detected outliers equals zero, then an array of length zero is returned.

One of the Compute methods must be invoked first before invoking this method. Otherwise, the method throws an InvalidOperationException exception.

**GetPsiWeights**

```csharp
public double[] GetPsiWeights()
```

**Description**

Returns the $\psi$ weights of the infinite order moving average form of the model.

**Returns**

A `double` array of length `nForecast` containing the $\psi$ weights of the infinite order moving average form of the optimum model for the outlier free series.

**Remarks**

Method `Forecast` must be invoked before this method is called. Otherwise, an InvalidOperationException exception is thrown.

**GetResiduals**

```csharp
public double[] GetResiduals()
```

**Description**

Returns the residuals.

**Returns**

A `double` array containing the residuals for the outlier and gap free series at the final parameter estimation point.

**Remarks**

Note that one of the Compute methods must be invoked first before invoking this method. Otherwise, `GetResiduals` throws an InvalidOperationException exception.

**SetDifferenceOrders**

```csharp
public void SetDifferenceOrders(int[] orders)
```

**Description**

Defines the orders of the periodic differences used in the determination of the optimum model.

**Parameter**

- `orders` – An `int` array containing all possible orders for each difference given in `periods`. All elements in `orders` must be non-negative.

  Default: `orders` is a one-element array with `orders[0] = 0`.

**SetPeriods**

```csharp
public void SetPeriods(int[] periods)
```

**Description**

Defines the periods used in the determination of the optimum model.
Parameter

periods – An int array containing all possible periods that can be applied to the original series after insertion of missing values. All elements of periods must be positive.

Default: periods is a one-element array with periods[0] = 1.

Example 1: Determination of an optimum AR\((p)\) model

This example uses time series LNU03327709 from the US Department of Labor, Bureau of Labor Statistics. It contains the unadjusted special unemployment rate, taken monthly from January 1994 through September 2005. The values 01/2004 - 03/2005 are used by class AutoARIMA for outlier detection and parameter estimation. In this example, method 1, without seasonal adjustment, is chosen to find an appropriate AR\((p)\) model. A forecast is done for the following six months and compared with the actual values 04/2005 - 09/2005.

```csharp
using System;
using Imsl.Stat;

class AutoARIMAEx1
{
    public static void Main(String[] args)
    {
        int nOutliers;
        double aic, RSE, constant;
        int[] optimumModel;
        int[,] outlierStatistics;
        double[] outlierForecast, ar, ma;
        double[] psiWeights, probabilityLimits;

        double[] x =
        { 12.8, 12.2, 11.9, 10.9, 10.6, 11.3, 11.1, 10.4, 10.0, 9.7, 9.7,
            9.7, 11.1, 10.5, 10.3, 9.8, 9.8, 10.4, 10.4, 10.0, 9.7, 9.3, 9.6,
            9.7, 10.8, 10.7, 10.3, 9.7, 9.5, 10.0, 10.0, 9.3, 9.0, 8.8, 8.9,
            9.2, 10.4, 10.0, 9.6, 9.0, 8.5, 9.2, 9.0, 8.6, 8.3, 7.9, 8.0,
            8.2, 9.3, 8.9, 8.9, 7.7, 7.6, 8.4, 8.5, 7.8, 7.6, 7.3, 7.2, 7.3,
            8.5, 8.2, 7.9, 7.4, 7.1, 7.9, 7.7, 7.2, 7.0, 6.7, 6.8, 6.9, 7.8,
            7.6, 7.4, 6.6, 6.8, 7.2, 7.2, 7.0, 6.6, 6.3, 6.8, 6.7, 8.1, 7.9,
            7.6, 7.1, 7.2, 8.2, 8.1, 8.1, 8.2, 8.7, 9.0, 9.3, 10.5, 10.1,
            9.9, 9.4, 9.2, 9.8, 9.9, 9.5, 9.0, 9.0, 9.4, 9.6, 11.0, 10.8,
            10.4, 9.8, 9.7, 10.6, 10.5, 10.0, 9.8, 9.5, 9.7, 9.6, 10.9, 10.3,
            10.4, 9.3, 9.3, 9.8, 9.8, 9.3, 8.9, 9.1, 9.1, 9.1, 10.2, 9.9, 9.4 }
;

double[] exactForecast = { 8.7, 8.6, 9.3, 8.8, 8.5 };

int[] times =
{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
    20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
    37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
    54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
    71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
```
AutoARIMA autoArima = new AutoARIMA(times, x);
autoArima.CriticalValue = 3.8;
autoArima.Compute(5);
autoArima.Forecast(6);

nOutliers = autoArima.NumberOfOutliers;
aic = autoArima.AIC;
optimumModel = autoArima.GetOptimumModelOrder();
outlierStatistics = autoArima.GetOutlierStatistics();
RSE = autoArima.ResidualStandardError;
outlierForecast = autoArima.GetForecast();
psiWeights = autoArima.GetPsiWeights();
probabilityLimits = autoArima.GetDeviations();
constant = autoArima.Constant;
ar = autoArima.GetAR();
ma = autoArima.GetMA();

Console.Out.WriteLine("\nMethod 1: Automatic AR model selection" + ", no differencing");
Console.Out.WriteLine("\nOptimum Model: p={0,1:d}, q={1,1:d}, s={2,1:d}, d={3,1:d}",
    optimumModel[0], optimumModel[1], optimumModel[2], optimumModel[3]);
Console.Out.WriteLine("\nNumber of outliers:{0,3:d}", nOutliers);

Console.Out.WriteLine(" Outlier statistics: ");
for (int i = 0; i < nOutliers; i++)
  Console.Out.WriteLine("\n{0,5:d}{1,8:d}", outlierStatistics[i, 0],
      outlierStatistics[i, 1]);

Console.Out.WriteLine("\nAIC:{0,12:f6}", aic);
Console.Out.WriteLine("\nRSE:{0,12:f6}", RSE);

Console.Out.WriteLine(" \nParameters");
for (int i = 0; i < ar.Length; i++)
  Console.Out.WriteLine(" ar[{0,1:d}]:{1,15:f6}", i, ar[i]);
for (int i = 0; i < ma.Length; i++)
  Console.Out.WriteLine(" ma[{0,1:d}]:{1,15:f6}", i, ma[i]);

Console.Out.WriteLine(" Exact forecast limits psi");
for (int i = 0; i < outlierForecast.Length; i++)
  Console.Out.WriteLine("\n{0,7:f4}{1,11:f4}{2,11:f4}{3,11:f4}",
      exactForecast[i], outlierForecast[i],
      probabilityLimits[i], psiWeights[i]);
Output

Method 1: Automatic AR model selection, no differencing

Optimum Model: p=5, q=0, s=1, d=0

Number of outliers: 7

Outlier statistics:

<table>
<thead>
<tr>
<th>Time</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>37</td>
<td>3</td>
</tr>
<tr>
<td>85</td>
<td>0</td>
</tr>
<tr>
<td>97</td>
<td>0</td>
</tr>
<tr>
<td>109</td>
<td>0</td>
</tr>
<tr>
<td>121</td>
<td>0</td>
</tr>
</tbody>
</table>

AIC: 371.104666
RSE: 0.359632

Parameters

constant: 0.097542
ar[0]: 0.891871
ar[1]: -0.123831
ar[2]: -0.138262
ar[3]: 0.135621
ar[4]: 0.224111

* * * Forecast Table * * *

<table>
<thead>
<tr>
<th>Exact</th>
<th>forecast</th>
<th>limits</th>
<th>psi</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.7000</td>
<td>9.1076</td>
<td>0.7049</td>
<td>0.8919</td>
</tr>
<tr>
<td>8.6000</td>
<td>9.0993</td>
<td>0.9445</td>
<td>0.6716</td>
</tr>
<tr>
<td>9.3000</td>
<td>9.4032</td>
<td>1.0565</td>
<td>0.3503</td>
</tr>
<tr>
<td>9.1000</td>
<td>9.5806</td>
<td>1.0849</td>
<td>0.2416</td>
</tr>
<tr>
<td>8.8000</td>
<td>9.5506</td>
<td>1.0982</td>
<td>0.4243</td>
</tr>
<tr>
<td>8.5000</td>
<td>9.3932</td>
<td>1.1382</td>
<td>0.5910</td>
</tr>
</tbody>
</table>

Example 2: Determination of an optimum ARIMA model via Grid search

This is the same as Example 1, except now class AutoARIMA uses Method 2 with a possible seasonal adjustment. As a result, the unadjusted model with \( p = 3, q = 2, s = 1, d = 0 \) is chosen as optimum.

```csharp
using System;
using Imsl.Stat;

public class AutoARIMAEx2
{
    public static void Main(String[] args)
    {
        int nOutliers;
        double aic, RSE, constant;
```
int[] optimumModel;
int[,] outlierStatistics;
double[] outlierForecast, ar, ma;
double[] psiWeights, probabilityLimits;
int[] arOrders = { 0, 1, 2, 3 };
int[] maOrders = { 0, 1, 2, 3 };
int[] periods = { 1, 2 };
int[] orders = { 0, 1, 2 };

double[] x =
{
12.8, 12.2, 11.9, 10.9, 10.6, 11.3, 11.1, 10.4, 10.0, 9.7, 9.7,
9.7, 11.1, 10.5, 10.3, 9.8, 9.8, 10.4, 10.4, 10.0, 9.7, 9.3, 9.6,
9.7, 10.8, 10.7, 10.3, 9.7, 9.5, 10.0, 10.0, 9.3, 9.0, 8.8, 8.9,
9.2, 10.4, 10.0, 9.6, 9.0, 8.5, 9.2, 9.0, 8.6, 8.3, 7.9, 8.0,
8.2, 9.3, 8.9, 8.9, 7.7, 7.6, 8.4, 8.5, 7.8, 7.6, 7.3, 7.2, 7.3,
8.5, 8.2, 7.9, 7.4, 7.1, 7.9, 7.7, 7.2, 7.0, 6.7, 6.8, 6.9, 7.8,
7.6, 7.4, 6.6, 6.8, 7.2, 7.2, 7.0, 6.6, 6.3, 6.8, 6.7, 8.1, 7.9,
7.6, 7.1, 7.2, 8.2, 8.1, 8.1, 8.2, 8.7, 9.0, 9.3, 10.5, 10.1,
9.9, 9.4, 9.2, 9.8, 9.9, 9.5, 9.0, 9.0, 9.4, 9.6, 11.0, 10.8,
10.4, 9.8, 9.7, 10.6, 10.5, 10.0, 9.8, 9.5, 9.7, 9.6, 10.9, 9.9,
10.3, 10.4, 9.3, 9.3, 9.8, 9.8, 9.3, 8.9, 9.1, 9.1, 9.1, 10.2, 9.9, 9.4
};

double[] exactForecast = { 8.7, 8.6, 9.3, 9.1, 8.8, 8.5 };

int[] times =
{
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103,
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
130, 131, 132, 133, 134, 135
};
AutoARIMA autoArima = new AutoARIMA(times, x);
autoArima.CriticalValue = 3.8;
autoArima.MaximumARLag = 5;
autoArima.SetPeriods(periods);
autoArima.SetDifferenceOrders(orders);
autoArima.Compute(arOrders, maOrders);
autoArima.Forecast(6);

nOutliers = autoArima.NumberOfOutliers;
aic = autoArima.AIC;
optimumModel = autoArima.GetOptimumModelOrder();
outlierStatistics = autoArima.GetOutlierStatistics();
RSE = autoArima.ResidualStandardError;
outlierForecast = autoArima.GetForecast();
psiWeights = autoArima.GetPsiWeights();
probabilityLimits = autoArima.GetDeviations();
constant = autoArima.Constant;
ar = autoArima.GetAR();
ma = autoArima.GetMA();

Console.Out.WriteLine("\nMethod 2: Grid search with differencing\n");
Console.Out.WriteLine();
Console.Out.WriteLine("Optimum Model: p={0,1:d}, q={1,1:d}, s={2,1:d}, d={3,1:d},
   optimumModel[0], optimumModel[1], optimumModel[2], optimumModel[3]);
Console.Out.WriteLine("\nNumber of outliers:{0,2:d}", nOutliers);
Console.Out.WriteLine();
Console.Out.WriteLine("Outlier statistics:\nTime Type\n");
for (int i = 0; i < nOutliers; i++)
   Console.Out.WriteLine("{0,5:d}{1,8:d}", outlierStatistics[i, 0],
   outlierStatistics[i, 1]);
Console.Out.WriteLine("AIC:{0,12:f6}", aic);
Console.Out.WriteLine("RSE:{0,12:f6}", RSE);
Console.Out.WriteLine();
Console.Out.WriteLine("Parameters\nconstant:{0,11:f6}", constant);
for (int i = 0; i < ar.Length; i++)
   Console.Out.WriteLine(" ar[{0,1:d}]: {1,13:f6}", i, ar[i]);
for (int i = 0; i < ma.Length; i++)
   Console.Out.WriteLine(" ma[{0,1:d}]: {1,13:f6}", i, ma[i]);

Output

Method 2: Grid search with differencing

Optimum Model: p=3, q=2, s=1, d=0

Number of outliers: 1

Outlier statistics:
Time Type
109 0

AIC:  408.108176
RSE:  0.412456

Parameters
constant:  0.554459
ar[0]:  1.940615
ar[1]:  -1.898025
ar[2]:  0.897791
ma[0]:  1.115803
Example 3: Specified ARIMA model

This example is the same as example 2 but now method 3 with the optimum model parameters p = 3, q = 2, s = 1, d = 0 from Example 2 is chosen for outlier detection and forecasting.

```csharp
using System;
using Imsl.Stat;

public class AutoARIMAEx3
{
    public static void Main(String[] args)
    {
        int nOutliers;
double aic, RSE, constant;
        int[] optimumModel;
        int[,] outlierStatistics;
        double[] outlierForecast, ar, ma;
        double[] psiWeights, probabilityLimits;

        double[] x =
        {
            12.8, 12.2, 11.9, 10.9, 10.6, 11.3, 11.1, 10.4, 10.0, 9.7, 9.7,
            9.7, 11.1, 10.5, 10.3, 9.8, 9.8, 10.4, 10.4, 10.0, 9.7, 9.3, 9.6,
            9.7, 10.8, 10.7, 10.3, 9.7, 9.5, 10.0, 10.0, 9.3, 9.0, 8.8, 8.9,
            9.2, 10.4, 10.0, 9.6, 9.0, 8.5, 9.2, 9.0, 8.6, 8.3, 7.9, 8.0,
            8.2, 9.3, 8.9, 8.9, 7.7, 7.6, 8.4, 8.5, 7.8, 7.6, 7.3, 7.2, 7.3,
            8.5, 8.2, 7.9, 7.4, 7.1, 7.9, 7.7, 7.2, 7.0, 6.7, 6.8, 6.9, 7.8,
            7.6, 7.4, 6.6, 6.8, 7.2, 7.2, 7.0, 6.6, 6.3, 6.8, 6.7, 8.1, 7.9,
            7.6, 7.1, 7.2, 8.2, 8.1, 8.1, 8.2, 8.7, 9.0, 9.3, 10.5, 10.1,
            9.9, 9.4, 9.2, 9.8, 9.9, 9.5, 9.0, 9.0, 9.4, 9.6, 11.0, 10.8,
            10.4, 9.8, 9.7, 10.6, 10.5, 10.0, 9.8, 9.5, 9.7, 9.6, 10.9, 10.3,
            10.4, 9.3, 9.3, 9.8, 9.8, 9.3, 8.9, 9.1, 9.1, 9.1, 10.2, 9.9, 9.4
        };

        double[] exactForecast = { 8.7, 8.6, 9.3, 8.1, 8.8, 8.5 };

        int[] times =
        {
            1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
            20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
            37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
            54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
            71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
```
AutoARIMA autoArima = new AutoARIMA(times, x);
autoArima.CriticalValue = 3.8;
autoArima.Compute(3, 2, 1, 0);
autoArima.Forecast(6);

nOutliers = autoArima.NumberOfOutliers;
aic = autoArima.AIC;
optimumModel = autoArima.GetOptimumModelOrder();
outlierStatistics = autoArima.GetOutlierStatistics();
RSE = autoArima.ResidualStandardError;
outlierForecast = autoArima.GetForecast();
psiWeights = autoArima.GetPsiWeights();
probabilityLimits = autoArima.GetDeviations();
constant = autoArima.Constant;
ar = autoArima.GetAR();
ma = autoArima.GetMA();

Console.Out.WriteLine("\nMethod 3: Specified ARIMA model\n");
Console.Out.WriteLine();
Console.Out.WriteLine("Optimum Model: p={0,1:d}, q={1,1:d}, s={2,1:d}, d={3,1:d},
          optimumModel[0], optimumModel[1], optimumModel[2], optimumModel[3]);
Console.Out.WriteLine("\nNumber of outliers:{0,2:d}\n", nOutliers);
Console.Out.WriteLine("Outlier statistics:\n");
Console.Out.WriteLine(" Time Type\n");
for (int i = 0; i < nOutliers; i++)
  Console.Out.WriteLine("{0,5:d}{1,8:d}", outlierStatistics[i, 0],
          outlierStatistics[i, 1]);
Console.Out.WriteLine("\nAIC:{0,13:f6}, aic\n", aic);
Console.Out.WriteLine("RSE:{0,13:f6}, RSE\n", RSE);
Console.Out.WriteLine();

Parameters\n
constant:{0,11:f6}, constant;
for (int i = 0; i < ar.Length; i++)
  Console.Out.WriteLine(" ar[{0,2:d}]:{1,13:f6}" , i, ar[i]);
for (int i = 0; i < ma.Length; i++)
  Console.Out.WriteLine(" ma[{0,2:d}]:{1,13:f6}" , i, ma[i]);

 ** ** Forecast Table ** **

Exact forecast limits psi\n
Output
Method 3: Specified ARIMA model

Time Series and Forecasting AutoARIMA • 963
Optimum Model: $p=3$, $q=2$, $s=1$, $d=0$

Number of outliers: 1

Outlier statistics:

<table>
<thead>
<tr>
<th>Time</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td>0</td>
</tr>
</tbody>
</table>

AIC: 408.108176
RSE: 0.412456

Parameters

- Constant: 0.554459
- $ar[0]$: 1.940615
- $ar[1]$: -1.898025
- $ar[2]$: 0.897791
- $ma[0]$: 1.115803
- $ma[1]$: -0.911902

* * * Forecast Table * * *

<table>
<thead>
<tr>
<th>Exact</th>
<th>forecast</th>
<th>limits</th>
<th>psi</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.7000</td>
<td>9.1085</td>
<td>0.8084</td>
<td>0.8248</td>
</tr>
<tr>
<td>8.6000</td>
<td>9.1715</td>
<td>1.0479</td>
<td>0.6145</td>
</tr>
<tr>
<td>9.3000</td>
<td>9.5039</td>
<td>1.1597</td>
<td>0.5248</td>
</tr>
<tr>
<td>9.1000</td>
<td>9.7677</td>
<td>1.2349</td>
<td>0.5926</td>
</tr>
<tr>
<td>8.8000</td>
<td>9.7051</td>
<td>1.3245</td>
<td>0.7056</td>
</tr>
<tr>
<td>8.5000</td>
<td>9.3817</td>
<td>1.4421</td>
<td>0.7157</td>
</tr>
</tbody>
</table>

AutoARIMA.InformationCriterion Enumeration

public enumeration Imsl.Stat.AutoARIMA.InformationCriterion

Indicates which information criterion is used in the optimum model search.

Fields

- **Akaike**
  public Imsl.Stat.AutoARIMA.InformationCriterion Akaike

  Description

  Indicates that Akaike’s information criterion (AIC) is used in the optimum model determination.

- **AkaikeCorrected**
  public Imsl.Stat.AutoARIMA.InformationCriterion AkaikeCorrected
Akaike’s corrected information criterion (AICC) is used in the optimum model determination.

Bayesian information criterion (BIC) is used in the optimum model determination.

Difference Class

Differences a seasonal or nonseasonal time series.

Class Difference performs \( m = \text{periods.Length} \) successive backward differences of period \( s_i = \text{periods}[i - 1] \) and order \( d_i = \text{orders}[i - 1] \) for \( i = 1, \ldots, m \) on the \( n = \text{z.Length} \) observations \( \{Z_t\} \) for \( t = 1, 2, \ldots, n \).

Consider the backward shift operator \( B \) given by

\[ B^k Z_t = Z_{t-k} \]

for all \( k \). Then, the backward difference operator with period \( s \) is defined by the following:

\[ \Delta_s Z_t = (1 - B^s) Z_t = Z_t - Z_{t-s} \text{ for } s \geq 0 \]

Note that \( B_s Z_t \) and \( \Delta_s Z_t \) are defined only for \( t = (s+1), \ldots, n \). Repeated differencing with period \( s \) is simply

\[ \Delta_s^d Z_t = (1 - B^s)^d Z_t = \sum_{j=0}^{d} \frac{d!}{j!(d-j)!} (-1)^j B^{sj} Z_t \]

where \( d \geq 0 \) is the order of differencing. Note that

\[ \Delta_s^d Z_t \]

is defined only for \( t = (sd+1), \ldots, n \).
The general difference formula used in the class `Difference` is given by

\[
W_T = \begin{cases} 
  \text{NaN} & \text{for } t = 1, \ldots, n_L \\
  \Delta_{s_1}^{d_1} \Delta_{s_2}^{d_2} \cdots \Delta_{s_m}^{d_m} Z_t & \text{for } t = n_L + 1, \ldots, n 
\end{cases}
\]

where \( n_L \) represents the number of observations “lost” because of differencing and NaN represents the missing value code. Note that

\[ n_L = \sum_j s_j d_j \]

A homogeneous, stationary time series can be arrived at by appropriately differencing a homogeneous, nonstationary time series (Box and Jenkins 1976, p. 85). Preliminary application of an appropriate transformation followed by differencing of a series can enable model identification and parameter estimation in the class of homogeneous stationary autoregressive moving average models.

### Property

**ObservationsLost**

```csharp
public int ObservationsLost { get; }
```

**Description**

Returns the number of observations lost because of differencing the time series. Note that the `Compute` method must be invoked first before invoking this method. Otherwise, the return value is 0.

**Property Value**

An `int` containing the number of observations lost because of differencing the time series `z`.

### Constructor

**Difference**

```csharp
public Difference()
```

**Description**

Constructor for `Difference`. 

---

966 • `Difference`  
IMSL C# Numerical Library
Methods

**Compute**

```java
public double[] Compute(double[] z, int[] periods)
```

**Description**

Computes a Difference series.

**Parameters**

- `z` – A double array containing the time series.
- `periods` – A int array containing the periods at which `z` is to be differenced.

**Returns**

A double array containing the differenced series.

**ExcludeFirst**

```java
public void ExcludeFirst(bool exclude)
```

**Description**

Excludes observations lost due to differencing.

**Parameter**

- `exclude` – A bool specifying whether or not to exclude lost observations due to differencing.

**Remarks**

If set to `true`, the observations lost due to differencing will be excluded. The differenced series will be the length of the number of observations minus the number of observations lost. If set to `false`, the observations lost due to differencing will be set to NaN (Not a number) and included in the differenced series. The default is to set the lost observations to NaN.

**SetOrders**

```java
public void SetOrders(int[] orders)
```

**Description**

Sets the orders for the Difference object.

**Parameter**

- `orders` – An int array of length equal to length of `periods`, containing the order of each difference given in `periods`.

**Remarks**

The elements of `orders` must be greater than or equal to 0.
Example 1: Difference

This example uses the Airline Data (Box and Jenkins 1976, p. 531) consisting of the monthly total number of international airline passengers from January 1949 through December 1960. Difference is used to compute ...

\[ W_t = \Delta_1 \Delta_{12} Z_t = (Z_t - Z_{t-12}) - (Z_{t-1} - Z_{t-13}) \]

for \( t = 14, 15, ..., 24 \).

```csharp
using System;
using Imsl.Stat;

public class DifferenceEx1
{
    public static void Main(String[] args)
    {
        int[] periods = new int[] {1, 12};
        int nLost;
        double[] z = new double[] {
            112.0, 118.0, 132.0, 129.0, 121.0,
            135.0, 148.0, 148.0, 136.0, 119.0,
            104.0, 118.0, 115.0, 126.0, 141.0,
            135.0, 125.0, 149.0, 170.0, 170.0,
            158.0, 133.0, 115.0, 141.0
        };

        Difference diff = new Difference();
        double[] output = diff.Compute(z, periods);
        nLost = diff.ObservationsLost;

        Console.Out.WriteLine("Observations Lost = " + nLost);
        for (int i = 0; i < output.Length; i++)
            Console.Out.WriteLine(output[i]);
    }
}
```

Output

Observations Lost = 13
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN

968 • Difference

IMSL C# Numerical Library
Example 2: Difference

This example uses the same data as Example 1. The first number of lost observations are excluded from W due to differencing, and the number of lost observations is also output.

```csharp
using System;
using Imsl.Stat;

public class DifferenceEx2
{
    public static void Main(String[] args)
    {
        int[] periods = new int[]{1, 12};
        int nLost;
        double[] z = new double[]{112.0, 118.0, 132.0, 129.0, 121.0,
                                  135.0, 148.0, 148.0, 136.0, 119.0,
                                  104.0, 118.0, 115.0, 126.0, 141.0,
                                  135.0, 125.0, 149.0, 170.0, 170.0,
                                  158.00, 133.0, 114.0, 140.0};

        Difference diff = new Difference();
        diff.ExcludeFirst(true);
        double[] output = diff.Compute(z, periods);
        nLost = diff.ObservationsLost;

        Console.Out.WriteLine
            ("The number of observation lost = " + nLost);
        for (int i = 0; i < output.Length; i++)
            Console.Out.WriteLine(output[i]);
    }
}
```

Output

The number of observation lost = 13

5
1
-3
-2
10
8
0
-8
-4
12
GARCH Class

public class Imsl.Stat.GARCH
Computes estimates of the parameters of a GARCH(p,q) model.
The Generalized Autoregressive Conditional Heteroskedastic (GARCH) model is defined as

\[ y_t = z_t \sigma_t \]

\[ \sigma_t^2 = \sigma^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2 + \sum_{i=1}^{q} \alpha_i y_{t-i}^2 \]

where \( z_t \)'s are independent and identically distributed standard normal random variables,

\[ \sigma > 0, \beta_i \geq 0, \alpha_i \geq 0 \]

and

\[ \sum_{i=1}^{p} \beta_i + \sum_{i=1}^{q} \alpha_i < 1 \]

The above model is denoted as GARCH(p, q). The \( p \) is the autoregressive lag and the \( q \) is the moving average lag. When \( \beta_i = 0, i = 1, 2, \ldots, p \), the above model reduces to ARCH(q) which was proposed by Engle (1982). The nonnegativity conditions on the parameters implied a nonnegative variance and the condition on the sum of the \( \beta_i \)'s and \( \alpha_i \)'s is required for wide sense stationarity.

In the empirical analysis of observed data, GARCH(1,1) or GARCH(1,2) models have often found to appropriately account for conditional heteroskedasticity (Palm 1996). This finding is similar to linear time series analysis based on ARMA models.

It is important to notice that for the above models positive and negative past values have a symmetric impact on the conditional variance. In practice, many series may have strong asymmetric influence on the conditional variance. To take into account this phenomena, Nelson (1991) put forward Exponential
GARCH (EGARCH). Lai (1998) proposed and studied some properties of a general class of models that extended linear relationship of the conditional variance in ARCH and GARCH into nonlinear fashion.

The maximum likelihood method is used in estimating the parameters in GARCH(p,q). The log-likelihood of the model for the observed series \( \{Y_t\} \) with length \( m \) is

\[
\log(L) = \frac{m}{2} \log(2\pi) - \frac{1}{2} \sum_{t=1}^{m} \frac{y_t^2}{\sigma_t^2} - \frac{1}{2} \sum_{t=1}^{m} \log \sigma_t^2,
\]

where \( \sigma_t^2 = \sigma^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2 + \sum_{i=1}^{q} \alpha_i y_{t-i}^2 \).

In the model, if \( q = 0 \), the model GARCH is singular such that the estimated Hessian matrix \( H \) is singular.

The initial values of the parameter array \( x[] \) entered in array \( xguess[] \) must satisfy certain constraints. The first element of \( xguess \) refers to sigma and must be greater than zero and less than MaxSigma. The remaining \( p+q \) initial values must each be greater than or equal to zero but less than one.

To guarantee stationarity in model fitting,

\[
\sum_{i=1}^{p+q} x(i) < 1,
\]

is checked internally. The initial values should be selected from the values between zero and one. The value of Akaike Information Criterion is computed by

\[
2 \times \log(L) + 2 \times (p + q + 1),
\]

where \( \log(L) \) is the value of the log-likelihood function at the estimated parameters.

In fitting the optimal model, the class Imsl.Math.MinConGenLin (p. 391), is modified to find the maximal likelihood estimates of the parameters in the model. Statistical inferences can be performed outside of the class GARCH based on the output of the log-likelihood function (LogLikelihood property), the Akaike Information Criterion (Akaike property), and the variance-covariance matrix (GetVarCovarMatrix method).

**Properties**

**Akaike**

public double Akaike {get; }
Description
Returns the value of Akaike Information Criterion evaluated at the estimated parameter array. Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is 0.

Property Value
A double scalar containing the value of Akaike Information Criterion evaluated at the estimated parameter array.

LogLikelihood
public double LogLikelihood {get; }

Description
Returns the value of Log-likelihood function evaluated at the estimated parameter array. Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is 0.

Property Value
A double scalar containing the value of Log-likelihood function evaluated at the estimated parameter array.

MaxSigma
public double MaxSigma {get; set; }

Description
The value of the upperbound on the first element (sigma) of the array of returned estimated coefficients.

Property Value
A double scalar containing the value of the upperbound on the first element (sigma) of the array of returned estimated coefficients.

Remarks
By default, MaxSigma= 10.

Sigma
public double Sigma {get; }

Description
Returns the estimated value of sigma squared. Note that the Compute method must be invoked first before invoking this method. Otherwise, the return value is NaN.

Property Value
A double scalar containing the estimated value of sigma squared.

Constructor
GARCH
public GARCH(int p, int q, double[] y, double[] xguess)
Description
Constructor for GARCH.

Parameters
p – A int scalar containing the number of autoregressive (AR) parameters.
q – A int scalar containing the number of moving average (MA) parameters.
y – A double array containing the observed time series data.
xguess – A double array of length p + q + 1 containing the initial values for the parameter array.

Exception
System.ArgumentException is thrown if the dimensions of y, and xguess are not consistent

Methods

Compute
public void Compute()

Description
Computes estimates of the parameters of a GARCH(p,q) model.

Exceptions
Imsl.Stat.ConstrInconsistentException is thrown if the equality constraints are inconsistent
Imsl.Stat.EqConstrInconsistentException is thrown if the equality constraints and the bounds on the variables are found to be inconsistent
Imsl.Stat.NoVectorXException is thrown if no vector X satisfies all of the constraints
Imsl.Stat.TooManyFunctionEvaluationsException is thrown if the number of function evaluations exceeded 1000
Imsl.Stat.VarsDeterminedException is thrown if the variables are determined by the equality constraints

GetAR
public double[] GetAR()

Description
Returns the estimated values of autoregressive (AR) parameters. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns
A double array of size p containing the estimated values of autoregressive (AR) parameters.

GetMA
public double[] GetMA()
Description
Returns the estimated values of moving average (MA) parameters. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns
A double array of size q containing the estimated values of moving average (MA) parameters.

GetVarCovarMatrix
public double[,] GetVarCovarMatrix()

Description
Returns the variance-covariance matrix. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns
A double matrix of size p + q + 1 by p + q + 1 containing the variance-covariance matrix.

GetX
public double[] GetX()

Description
Returns the estimated parameter array, x. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns
A double array of size p + q + 1 containing the estimated values of sigma squared, the AR parameters, and the MA parameters.

Example: GARCH
The data for this example are generated to follow a GARCH(p,q) process by using a random number generation function sgarch. The data set is analyzed and estimates of sigma, the AR parameters, and the MA parameters are returned. The values of the Log-likelihood function and the Akaike Information Criterion are returned.

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class GARCHEx1
{
    static private void sgarch(int p, int q, int m, double[] x, double[] y, double[] z, double[] y0, double[] sigma)
    {
        int i, j, l;
        double s1, s2, s3;
        rand.Multiplier = 16807;
```
for (i = 0; i < m + 1000; i++)
    z[i] = rand.NextNormal();

l = System.Math.Max(p, q);
l = System.Math.Max(l, 1);
for (i = 0; i < l; i++)
    y0[i] = z[i] * x[0];

/* COMPUTE THE INITIAL VALUE OF SIGMA */
s3 = 0.0;
if (System.Math.Max(p, q) >= 1)
{
    for (i = 1; i < (p + q + 1); i++)
        s3 += x[i];
}
for (i = 0; i < l; i++)
    sigma[i] = x[0] / (1.0 - s3);
for (i = l; i < (m + 1000); i++)
{
    s1 = 0.0;
    s2 = 0.0;
    if (q >= 1)
    {
        for (j = 0; j < q; j++)
            s1 += x[j + 1] * y0[i - j - 1] * y0[i - j - 1];
    }
    if (p >= 1)
    {
        for (j = 0; j < p; j++)
            s2 += x[q + 1 + j] * sigma[i - j - 1];
    }
    sigma[i] = x[0] + s1 + s2;
    y0[i] = z[i] * Math.Sqrt(sigma[i]);
}
/*
 * DISCARD THE FIRST 1000 SIMULATED OBSERVATIONS
 */
for (i = 0; i < m; i++)
    y[i] = y0[1000 + i];
return ;

public static void Main(String[] args)
{
    int n, p, q, m;
double[] x = new double[]{1.3, 0.2, 0.3, 0.4};
double[] xguess = new double[]{1.0, 0.1, 0.2, 0.3};
double[] y = new double[1000];
double[] wk1 = new double[2000];
double[] wk2 = new double[2000];
double[] wk3 = new double[2000];

m = 1000;
p = 2;
q = 1;
\[ n = p + q + 1; \]
\[ \text{sgarch}(p, q, m, x, y, wk1, wk2, wk3); \]
\[ \text{GARCH} \text{ garch = new GARCH}(p, q, y, xguess); \]
\[ \text{garch.Compute();} \]
\[ \text{Console.Out.WriteLine} \]
\[ ("\text{Sigma estimate is } \text{garch.Sigma.ToString("0.000")}); \]
\[ \text{Console.Out.WriteLine();} \]
\[ \text{new PrintMatrix("AR estimate is ").Print(garch.GetAR());} \]
\[ \text{new PrintMatrix("MR estimate is ").Print(garch.GetMA());} \]
\[ \text{Console.Out.WriteLine("Log-likelihood function value is } \text{garch.LogLikelihood.ToString("0.000")});} \]
\[ \text{Console.Out.WriteLine("Akaike Information Criterion value is } \text{garch.Akaike.ToString("0.000")});} \]

\[ \text{Output} \]

Sigma estimate is 1.692

AR estimate is

\[
\begin{array}{ccc}
0 & 0.24496841351061 \\
1 & 0.337228450714669 \\
\end{array}
\]

MR estimate is

\[
\begin{array}{ccc}
0 & 0.30959276087191 \\
\end{array}
\]

Log-likelihood function value is -2707.073
Akaike Information Criterion value is 5422.146

\[ \text{KalmanFilter Class} \]

\[ \text{public class Imsl.Stat.KalmanFilter} \]

Performs Kalman filtering and evaluates the likelihood function for the state-space model.

Class KalmanFilter is based on a recursive algorithm given by Kalman (1960), which has come to be known as the KalmanFilter. The underlying model is known as the state-space model. The model is specified stage by stage where the stages generally correspond to time points at which the observations become available. KalmanFilter avoids many of the computations and storage requirements that would be necessary if one were to process all the data at the end of each stage in order to estimate the state vector. This is accomplished by using previous computations and retaining in storage only those items essential for processing of future observations.

The notation used here follows that of Sallas and Harville (1981). Let \( y_k \) (input in \( y \) using method
Update) be the \( n_k \times 1 \) vector of observations that become available at time \( k \). The subscript \( k \) is used here rather than \( t \), which is more customary in time series, to emphasize that the model is expressed in stages \( k = 1, 2, \ldots \) and that these stages need not correspond to equally spaced time points. In fact, they need not correspond to time points of any kind. The observation equation for the state-space model is

\[
y_k = Z_k b_k + e_k \quad k = 1, 2, \ldots
\]

Here, \( Z_k \) (input in \( z \) using method update) is an \( n_k \times q \) known matrix and \( b_k \) is the \( q \times 1 \) state vector. The state vector \( b_k \) is allowed to change with time in accordance with the state equation

\[
b_{k+1} = T_{k+1} b_k + w_{k+1} \quad k = 1, 2, \ldots
\]

starting with \( b_1 = \mu_1 + w_1 \).

The change in the state vector from time \( k \) to \( k + 1 \) is explained in part by the transition matrix \( T_{k+1} \) (the identity matrix by default, or optionally using method SetTransitionMatrix), which is assumed known. It is assumed that the \( q \)-dimensional \( w_k \)'s \((k = 1, 2, \ldots)\) are independently distributed multivariate normal with mean vector 0 and variance-covariance matrix \( \sigma^2 Q_k \), that the \( n_k \)-dimensional \( e_k \)'s \((k = 1, 2, \ldots)\) are independently distributed multivariate normal with mean vector 0 and variance-covariance matrix \( \sigma^2 R_k \), and that the \( w_k \)'s and \( e_k \)'s are independent of each other. Here, \( \mu_1 \) is the mean of \( b_1 \) and is assumed known, \( \sigma^2 \) is an unknown positive scalar. \( Q_{k+1} \) (input in \( Q \)) and \( R_k \) (input in \( R \)) are assumed known.

Denote the estimator of the realization of the state vector \( b_k \) given the observations \( y_1, y_2, \ldots, y_j \) by

\[
\hat{\beta}_{k|j}
\]

By definition, the mean squared error matrix for

\[
\hat{\beta}_{k|j}
\]

is

\[
\sigma^2 C_{k|j} = E(\hat{\beta}_{k|j} - b_k)(\hat{\beta}_{k|j} - b_k)^T
\]

At the time of the \( k \)-th invocation, we have

\[
\hat{\beta}_{k|k-1}
\]
and

\( C_{k|k-1} \), which were computed from the \( k-1 \)-st invocation, input in \( b \) and \( \text{covb} \), respectively. During the \( k \)-th invocation, \texttt{KalmanFilter} computes the filtered estimate

\[
\hat{\beta}_{k|k}
\]

along with \( C_{k|k} \). These quantities are given by the update equations:

\[
\hat{\beta}_{k|k} = \hat{\beta}_{k|k-1} + C_{k|k-1} Z_k^T H_k^{-1} v_k
\]

\[
C_{k|k} = C_{k|k-1} - C_{k|k-1} Z_k^T H_k^{-1} Z_k C_{k|k-1}
\]

where

\[
v_k = y_k - Z_k \hat{\beta}_{k|k-1}
\]

and where

\[
H_k = R_k + Z_k C_{k|k-1} Z_k^T
\]

Here, \( v_k \) (stored in \( v \)) is the one-step-ahead prediction error, and \( \sigma^2 H_k \) is the variance-covariance matrix for \( v_k \). \( H_k \) is stored in \( \text{covv} \). The "start-up values" needed on the first invocation of \texttt{KalmanFilter} are

\[
\hat{\beta}_{1|0} = \mu_1
\]

and \( C_{1|0} = Q_t \) input via \( b \) and \( \text{covb} \), respectively. Computations for the \( k \)-th invocation are completed by \texttt{KalmanFilter} computing the one-step-ahead estimate

\[
\hat{\beta}_{k+1|k}
\]

along with \( C_{k+1|k} \) given by the prediction equations:

\[
\hat{\beta}_{k+1|k} = T_{k+1} \hat{\beta}_{k|k}
\]
\[ C_{k+1|k} = T_{k+1|k} C_{k|k} T_{k+1|k}^T + Q_{k+1|k} \]

If both the filtered estimates and one-step-ahead estimates are needed by the user at each time point, `KalmanFilter` can be used twice for each time point-first without methods `SetTransitionMatrix` and `SetQ` to produce

\[ \hat{\beta}_{k|k} \]

and \( C_{k|k} \), and second without method `Update` to produce

\[ \hat{\beta}_{k+1|k} \]

and \( C_{k+1|k} \). (Without methods `SetTransitionMatrix` and `SetQ`, the prediction equations are skipped. Without method `Update`, the update equations are skipped.)

Often, one desires the estimate of the state vector more than one-step-ahead, i.e., an estimate of

\[ \hat{\beta}_{k|j} \]

is needed where \( k > j + 1 \). At time \( j \), `KalmanFilter` is invoked with method `Update` to compute

\[ \hat{\beta}_{j+1|j} \]

Subsequent invocations of `KalmanFilter` without method `Update` can compute

\[ \hat{\beta}_{j+2|j}, \hat{\beta}_{j+3|j}, \ldots, \hat{\beta}_{k|j} \]

 Computations for

\[ \hat{\beta}_{k|j} \]

and \( C_{k|j} \) assume the variance-covariance matrices of the errors in the observation equation and state equation are known up to an unknown positive scalar multiplier, \( \sigma^2 \). The maximum likelihood estimate of \( \sigma^2 \) based on the observations \( y_1, y_2, \ldots, y_m \), is given by

\[ \hat{\sigma}^2 = SS/N \]
where
\[ N = \sum_{k=1}^{m} n_k \quad \text{and} \quad SS = \sum_{k=1}^{m} v_k^T H_k^{-1} v_k \]

\( N \) and \( SS \) are the input/output arguments \( n \) and \( \text{sumOfSquares} \).

If \( \sigma^2 \) is known, the \( R_k \)s and \( Q_k \)s can be input as the variance-covariance matrices exactly. The earlier discussion is then simplified by letting \( \sigma^2 = 1 \).

In practice, the matrices \( T_k \), \( Q_k \), and \( R_k \) are generally not completely known. They may be known functions of an unknown parameter vector \( \theta \). In this case, \( \text{KalmanFilter} \) can be used in conjunction with an optimization class (see class \text{MinUnconMultiVar}, IMSL C# Library Math namespace), to obtain a maximum likelihood estimate of \( \theta \). The natural logarithm of the likelihood function for \( y_1, y_2, \ldots, y_m \) differs by no more than an additive constant from
\[
L(\theta, \sigma^2; y_1, y_2, \ldots, y_m) = -\frac{1}{2} N \ln \sigma^2 - \frac{1}{2} \sum_{k=1}^{m} \ln(|H_k|) - \frac{1}{2} \sigma^2 \sum_{k=1}^{m} v_k^T H_k^{-1} v_k
\]

(Harvey 1981, page 14, equation 2.21).

Here,
\[
\sum_{k=1}^{m} \ln(|H_k|)
\]

(stored in \text{logDeterminant}) is the natural logarithm of the determinant of \( V \) where \( \sigma^2 V \) is the variance-covariance matrix of the observations.

Minimization of \(-2L(\theta, \sigma^2; y_1, y_2, \ldots, y_m)\) over all \( \theta \) and \( \sigma^2 \) produces maximum likelihood estimates. Equivalently, minimization of \(-2L_c(\theta; y_1, y_2, \ldots, y_m)\) where
\[
L_c(\theta; y_1, y_2, \ldots, y_m) = -\frac{1}{2} N \ln \left( \frac{SS}{N} \right) - \frac{1}{2} \sum_{k=1}^{m} \ln(|H_k|)
\]

produces maximum likelihood estimates
\[
\hat{\theta} \quad \text{and} \quad \hat{\sigma}^2 = \frac{SS}{N}
\]

Minimization of \(-2L_c(\theta; y_1, y_2, \ldots, y_m)\) instead of \(-2L(\theta, \sigma^2; y_1, y_2, \ldots, y_m)\), reduces the dimension of the minimization problem by one. The two optimization problems are equivalent since
\[
\hat{\sigma}^2(\theta) = \frac{SS(\theta)}{N}
\]
minimizes $-2L(\theta, \sigma^2; y_1, y_2, \ldots, y_m)$ for all $\theta$, consequently,

$$\hat{\sigma}^2(\theta)$$

can be substituted for $\sigma^2$ in $L(\theta, \sigma^2; y_1, y_2, \ldots, y_m)$ to give a function that differs by no more than an additive constant from $L_c(\theta; y_1, y_2, \ldots, y_m)$.

The earlier discussion assumed $H_k$ to be nonsingular. If $H_k$ is singular, a modification for singular distributions described by Rao (1973, pages 527-528) is used. The necessary changes in the preceding discussion are as follows:

1. Replace $H_k^{-1}$ by a generalized inverse.
2. Replace $\det(H_k)$ by the product of the nonzero eigenvalues of $H_k$.
3. Replace $N$ by $\sum_{i=1}^{m} \text{rank}(H_k)$


### Properties

**LogDeterminant**

```csharp
public double LogDeterminant {get; }
```

**Description**

Returns the natural log of the product of the nonzero eigenvalues of $P$ where $P * \sigma^2$ is the variance-covariance matrix of the observations.

**Property Value**

A `double` scalar containing the natural log of the product of the nonzero eigenvalues of $P$ where $P * \sigma^2$ is the variance-covariance matrix of the observations.

**Remarks**

In the usual case when $P$ is nonsingular, LogDeterminant is the natural log of the determinant of $P$.

```csharp
public int Rank {get; }
```

**Description**

Returns the rank of the variance-covariance matrix for all the observations.
**Property Value**
A `int` scalar containing the rank of the variance-covariance matrix for all the observations.

**SumOfSquares**
public double SumOfSquares {get; }

**Description**
Returns the generalized sum of squares.

**Property Value**
A `double` scalar containing the generalized sum of squares.

**Remarks**
The estimate of \( \sigma^2 \) is given by \( \text{sumOfSquares} / n \).

**Tolerance**
public double Tolerance {get; set; }

**Description**
The tolerance used in determining linear dependence.

**Property Value**
A `double` scalar containing the tolerance used in determining linear dependence.

**Remarks**
By default, \( \text{Tolerance} = 100.0 \times 2.2204460492503131 \times 10^{-16} \).

**Constructor**

**KalmanFilter**
public KalmanFilter(double[] b, double[,] covb, int rank, double sumOfSquares, double logDeterminant)

**Description**
Constructor for `KalmanFilter`.

**Parameters**
- `b` – A `double` array containing the estimated state vector.
- `covb` – A `double` matrix of size `b.Length` by `b.Length` such that `covb * \sigma^2` is the mean squared error matrix for `b`.
- `rank` – An `int` scalar containing the rank of the variance-covariance matrix for all the observations.
- `sumOfSquares` – A `double` scalar containing the generalized sum of squares.
- `logDeterminant` – A `double` scalar containing the natural log of the product of the nonzero eigenvalues of `P` where `P * \sigma^2` is the variance-covariance matrix of the observations.
Remarks

b is the estimated state vector at time k given the observations through time k-1.

Exception

System.ArgumentException is thrown if the dimensions of b, and covb are not consistent.

Methods

Filter

public void Filter()

Description

Performs Kalman filtering and evaluates the likelihood function for the state-space model.

GetCovB

public double[,] GetCovB()

Description

Returns the mean squared error matrix for b divided by sigma squared.

Returns

A double matrix of size b.Length by b.Length such that covb * σ^2 is the mean squared error matrix for b.

GetCovV

public double[,] GetCovV()

Description

Returns the variance-covariance matrix of v divided by sigma squared.

Returns

A double matrix containing a y.Length by y.Length matrix such that covv * σ^2 is the variance-covariance matrix of v.

GetPredictionError

public double[] GetPredictionError()

Description

Returns the one-step-ahead prediction error.

Returns

A double array of size y.Length containing the one-step-ahead prediction error.

GetStateVector

public double[] GetStateVector()
Description
Returns the estimated state vector at time k + 1 given the observations through time k.

Returns
A double array containing the estimated state vector at time k + 1 given the observations through time k.

ResetQ
public void ResetQ()

Description
Removes the Q matrix.

ResetTransitionMatrix
public void ResetTransitionMatrix()

Description
Removes the transition matrix.

ResetUpdate
public void ResetUpdate()

Description
Do not perform computation of the update equations.

SetQ
public void SetQ(double[,] q)

Description
Sets the Q matrix.

Parameter
q – A double matrix containing the b.Length by b.Length matrix such that q * σ^2 is the variance-covariance matrix of the error vector in the state equation.

Remarks
By default, there is no error term in the state equation.

SetTransitionMatrix
public void SetTransitionMatrix(double[,] t)

Description
Sets the transition matrix.

Parameter
t – A double matrix containing the b.Length by b.Length transition matrix in the state equation.
Remarks
By default, \( t = \text{identity matrix} \).

Update
public void Update(double[] y, double[,] z, double[,] r)

Description
Performs computation of the update equations.

Parameters
- \( y \) – A double array containing the observations.
- \( z \) – A double matrix containing the \( y.\text{Length} \times b.\text{Length} \) matrix relating the observations to the state vector in the observation equation.
- \( r \) – A double matrix containing the \( y.\text{Length} \times y.\text{Length} \) matrix such that \( r \times \sigma^2 \) is the variance-covariance matrix of errors in the observation equation.

Remarks
\( \sigma^2 \) is a positive unknown scalar. Only elements in the upper triangle of \( r \) are referenced.

Example 1: Kalman Filter
KalmanFilter is used to compute the filtered estimates and one-step-ahead estimates for a scalar problem discussed by Harvey (1981, pages 116-117). The observation equation and state equation are given by

\[
y_k = b_k + e_k
\]

\[
b_{k+1} = b_k + w_{k+1}
\]

\( k = 1, 2, 3, 4 \)

where the \( e_k \)'s are identically and independently distributed normal with mean 0 and variance \( \sigma^2 \), the \( w_k \)'s are identically and independently distributed normal with mean 0 and variance \( 4\sigma^2 \), and \( b_1 \) is distributed normal with mean 4 and variance \( 16\sigma^2 \). Two KalmanFilter objects are needed for each time point in order to compute the filtered estimate and the one-step-ahead estimate. The first object does not use the methods SetTransitionMatrix and SetQ so that the prediction equations are skipped in the computations. The update equations are skipped in the computations in the second object.

This example also computes the one-step-ahead prediction errors. Harvey (1981, page 117) contains a misprint for the value \( v_4 \) that he gives as 1.197. The correct value of \( v_4 = 1.003 \) is computed by KalmanFilter.

using System;
using Imsl.Stat;

public class KalmanFilterEx1
{

}
private static readonly String format = 
"{0}/\{1}\{2:0.000}\{3:0.000}\{4:0.000}\{5:0.000}" + 
"\{6:0.000}\{7:0.000}\{8:0.000}\{9:0.000};"

public static void Main(String[] args) {
    int nobs = 4;
    int rank = 0;
    double logDeterminant = 0.0;
    double ss = 0.0;
    double[] b = new double[]{4};
    double[,] covb = new double[1,1]{16};
    double[,] q = {{4}};
    double[,] r = {{1}};
    double[,] t = {{1}};
    double[,] z = {{1}};
    double[] ydata = new double[]{4.4, 4.0, 3.5, 4.6};

    System.Object[] argFormat =
        new System.Object[]{"k", "j", "b", "cov(b)", "rank", "ss", 
        "ln(det)", "v", "cov(v)"};
    Console.Out.WriteLine(String.Format(format, argFormat));

    KalmanFilter kalman =
        new KalmanFilter(b, covb, rank, ss, logDeterminant);

    for (int i = 0; i < nobs; i++)
    {
        double[] y = new double[]{ydata[i]};
        kalman.Update(y, z, r);
        kalman.Filter();
        double[] v = kalman.GetPredictionError();
        double[,] covv = kalman.GetCovV();
        argFormat[0] = i;
        argFormat[1] = i;
        argFormat[2] = kalman.GetStateVector()[0];
        argFormat[3] = kalman.GetCovB()[0,0];
        argFormat[7] = v[0];
        argFormat[8] = covv[0,0];
        Console.Out.WriteLine(String.Format(format, argFormat));
        kalman.ResetUpdate();

        kalman.SetTransitionMatrix(t);
        kalman.SetQ(q);
        kalman.Filter();
        argFormat[0] = i + 1;
        argFormat[1] = i;
        argFormat[2] = kalman.GetStateVector()[0];
        argFormat[3] = kalman.GetCovB()[0,0];
        argFormat[7] = v[0];
argFormat[8] = covv[0,0];
Console.Out.WriteLine(String.Format(format, argFormat));
kalman.ResetTransitionMatrix();
kalman.ResetQ();

Output

<table>
<thead>
<tr>
<th>k/j</th>
<th>b</th>
<th>cov(b)</th>
<th>rank</th>
<th>ss</th>
<th>ln(det)</th>
<th>v</th>
<th>cov(v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/0</td>
<td>4.376</td>
<td>0.941</td>
<td>1</td>
<td>0.009</td>
<td>2.833</td>
<td>0.400</td>
<td>17.000</td>
</tr>
<tr>
<td>1/0</td>
<td>4.376</td>
<td>4.941</td>
<td>1</td>
<td>0.009</td>
<td>2.833</td>
<td>0.400</td>
<td>17.000</td>
</tr>
<tr>
<td>1/1</td>
<td>4.063</td>
<td>0.832</td>
<td>2</td>
<td>0.033</td>
<td>4.615</td>
<td>-0.376</td>
<td>5.941</td>
</tr>
<tr>
<td>2/1</td>
<td>4.063</td>
<td>4.832</td>
<td>2</td>
<td>0.033</td>
<td>4.615</td>
<td>-0.376</td>
<td>5.941</td>
</tr>
<tr>
<td>2/2</td>
<td>3.597</td>
<td>0.829</td>
<td>3</td>
<td>0.088</td>
<td>6.378</td>
<td>-0.563</td>
<td>5.832</td>
</tr>
<tr>
<td>3/2</td>
<td>3.597</td>
<td>4.829</td>
<td>3</td>
<td>0.088</td>
<td>6.378</td>
<td>-0.563</td>
<td>5.832</td>
</tr>
<tr>
<td>3/3</td>
<td>4.428</td>
<td>0.828</td>
<td>4</td>
<td>0.260</td>
<td>8.141</td>
<td>1.003</td>
<td>5.829</td>
</tr>
<tr>
<td>4/3</td>
<td>4.428</td>
<td>4.828</td>
<td>4</td>
<td>0.260</td>
<td>8.141</td>
<td>1.003</td>
<td>5.829</td>
</tr>
</tbody>
</table>

Example 2: Kalman Filter Maximum Likelihood Estimate

KalmanFilter is used with the MinUnconMultivar class to find a maximum likelihood estimate of the parameter \( \theta \) in an MA(1) time series represented by \( y_k = \epsilon_k - \theta \epsilon_{k-1} \). The input is 200 observations from an MA(1) time series with \( \theta = 0.5 \) and \( \sigma^2 = 1 \). The MA(1) time series is cast as a state-space model of the following form (see Harvey 1981, pages 103-104, 112):

\[
y_k = (1 \ 0) b_k
\]

\[
b_k = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} b_{k-1} + w_k
\]

where the two-dimensional \( w_k \)'s are independently distributed bivariate normal with mean 0 and variance \( \sigma^2 Q_k \) and

\[
Q_1 = \begin{pmatrix} 1 + \theta^2 & -\theta \\ -\theta & \theta^2 \end{pmatrix}
\]

\[
Q_k = \begin{pmatrix} 1 & -\theta \\ -\theta & \theta^2 \end{pmatrix}, \quad k = 2, 3, ..., 200
\]
public double F(double[] theta)
{
    int nobs = 200;
    int rank = 0;
    double ss = 0.0;
    double logDeterminant = 0.0;
    double res;
    double[,] covb = new double[2, 2];
    double[,] q = new double[2, 2];
    double[,] r = {{0.0, 0.0}};
    double[] b = {0.0, 0.0};
    double[,] z = {{1.0, 0.0}};
    double[,] t = {{0.0, 1.0},
                   {0.0, 0.0}};
    double[] y = new double[1];
    double[] ydata = {0.057466, -0.459067, 1.236446, -1.864825, 0.951507,
                       -1.489387, -0.864401, 0.302984, -0.376017, 0.610208,
                       -1.701583, 2.271794, -0.917110, 0.582576, 1.018681,
                       0.107443, -0.557858, 0.502818, 0.858902, -1.417659,
                       0.839981, 0.047021, 0.404448, 0.383749, -0.383227,
                       1.179607, -1.154339, 1.927541, 0.996344, 1.019415,
                       -0.816815, -0.100467, -0.625334, -0.068065, -1.891685,
                       0.657241, -1.070823, 1.023510, 2.672653, -2.141434,
                       -1.232266, 0.925311, -0.201665, -1.325580, -0.086926,
                       -0.647157, 1.314749, -0.085708, 1.430485, 0.304040,
                       0.305559, 1.669671, 1.004800, -1.678350, 0.631133,
                       0.502284, 0.247378, -1.345484, 0.994982, 1.145546,
                       -1.248822, 0.616784, -2.127822, 2.264872, -1.590343,
                       0.365786, -1.056652, 0.969750, 1.028274, 0.332050,
                       0.430686, 0.364653, 0.482446, -0.303248, 1.581931,
                       -0.140942, -0.265280, -0.939284, 0.464963, -0.778145,
                       0.583466, -1.113080, -0.009839, 1.580189, -1.116377,
                       1.744513, -0.296106, 0.332944, -0.228859, 1.101747,
                       0.772369, -1.608111, 2.671822, -0.504800, -1.647797,
                       -0.596313, 0.845472, 0.507869, 0.833777, -0.460099,
                       0.416891, -1.139069, 0.159028, -0.193971, -0.154656,
                       1.720997, -0.403189, 0.400026, 0.285921, -1.914338,
                       1.296864, 1.426898, -0.426181, 0.255961, -1.790193,
                       0.721048, 1.150173, -0.980366, 0.940958, 0.313898,
                       0.505735, -1.058126, 0.111918, 0.185493, -0.296146,
                       -0.104457, 1.151733, 0.683025, -0.714269, -0.787972,
                       -1.277062, 1.378816, -0.658115, -0.259860, -2.614008,
                       2.251646, -0.006773, -0.738467, -0.260685, 1.896505,
                       -0.094919, 0.089954, -1.627679, -1.675018, 0.896835,
                       0.498690, -0.368775, 0.131849, -2.060292, 0.272666,
                       2.115804, -1.323451, 0.557106, -0.602031, 1.424524,
                       -0.107996, -1.580671, 0.672012, -1.668931, 2.474710,
                       -1.471518, 1.780317, -0.419588, 2.008474, -1.246855,
                       0.231161, 0.706104, -0.474320, -0.705431, 0.593588,
                       -2.469494, 2.024374, 0.849572, -2.410618, 2.812321,
                       -1.066520, -0.539768, -0.067784, 1.970878, 0.592879,
                       -0.184623, -1.403912, -0.995537, 1.727320, -0.313251,
                       0.472437, -0.241800, -0.875680, -0.159557, 0.508238,
                       -0.116888, -0.981759, -0.472451, 0.847273, -1.713030,
                       2.010192, -0.981891, 1.190901, 0.453348, -0.743333
};
if (Math.Abs(theta[0]) > 1.0)
{
    res = 1.0e10;
}
else
{
    q[0, 0] = 1.0;
    q[0, 1] = -theta[0];
    q[1, 0] = -theta[0];
    q[1, 1] = theta[0] * theta[0];

    covb[0, 0] = 1.0 + theta[0] * theta[0];
    covb[0, 1] = -theta[0];
    covb[1, 0] = -theta[0];
    covb[1, 1] = theta[0] * theta[0];

    KalmanFilter kalman = new KalmanFilter(b, covb, rank, ss, logDeterminant);

    for (int i = 0; i < nobs; i++)
    {
        y[0] = ydata[i];
        kalman.Update(y, z, x);
        kalman.SetTransitionMatrix(t);
        kalman.SetQ(q);
        kalman.Filter();
    }
    ss = kalman.SumOfSquares;
    logDeterminant = kalman.LogDeterminant;
    rank = kalman.Rank;
    res = rank * Math.Log(ss / rank) + logDeterminant;
}
return (res);
}

public static void Main(String[] args)
{
    MinUnconMultiVar solver = new MinUnconMultiVar(1);
    double[] x = solver.ComputeMin(new KalmanFilterEx2());
    Console.Out.WriteLine("Maximum likelihood estimate, THETA = \{0\}", x[0]);
}

Output
Maximum likelihood estimate, THETA = 0.452940213528753
Chapter 19: Multivariate Analysis

Types

class ClusterKMeans ................................................................. 993
class Dissimilarities ................................................................. 1002
enumeration Dissimilarities.Method .......................................... 1007
enumeration Dissimilarities.Scaling ........................................... 1009
class ClusterHierarchical ............................................................ 1009
enumeration ClusterHierarchical.Linkage .................................... 1018
enumeration ClusterHierarchical.Transformation .......................... 1019
class FactorAnalysis ................................................................. 1020
tenumeration FactorAnalysis.MatrixType ..................................... 1037
tenumeration FactorAnalysis.Model ............................................. 1037
class DiscriminantAnalysis .......................................................... 1038
enumeration DiscriminantAnalysis.Discrimination ....................... 1060
tenumeration DiscriminantAnalysis.CovarianceMatrix .................... 1061
tenumeration DiscriminantAnalysis.Classification ......................... 1061
enumeration DiscriminantAnalysis.PriorProbabilities ..................... 1062

Usage Notes

Cluster Analysis

ClusterKMeans performs a K-means cluster analysis. Basic K-means clustering attempts to find a clustering that minimizes the within-cluster sums-of-squares. In this method of clustering the data, matrix X is grouped so that each observation (row in X) is assigned to one of a fixed number, K, of clusters. The sum of the squared difference of each observation about its assigned cluster’s mean is used as the criterion for assignment. In the basic algorithm, observations are transferred from one cluster or another when doing so decreases the within-cluster sums-of-squared differences. When no transfer occurs in a pass through the entire data set, the algorithm stops. ClusterKMeans is one implementation of the basic algorithm.

The usual course of events in K-means cluster analysis is to use ClusterKMeans to obtain the optimal clustering. The clustering is then evaluated by functions described in “Basic Statistics”, and/or other chapters in this manual. Often, K-means clustering with more than one value of K is performed, and the
value of K that best fits the data is used.

Clustering can be performed either on observations or variables. The discussion of the function `ClusterKMeans` assumes the clustering is to be performed on the observations, which correspond to the rows of the input data matrix. If variables, rather than observations, are to be clustered, the data matrix should first be transposed. In the documentation for `ClusterKMeans`, the words “observation” and “variable” are interchangeable.

**Principal Components**

The idea in principal components is to find a small number of linear combinations of the original variables that maximize the variance accounted for in the original data. This amounts to an eigensystem analysis of the covariance (or correlation) matrix. In addition to the eigensystem analysis, when the principal component model is used, `FactorAnalysis` computes standard errors for the eigenvalues. Correlations of the original variables with the principal component scores also are computed.

**Factor Analysis**

Factor analysis and principal component analysis, while quite different in assumptions, often serve the same ends. Unlike principal components in which linear combinations yielding the highest possible variances are obtained, factor analysis generally obtains linear combinations of the observed variables according to a model relating the observed variable to hypothesized underlying factors, plus a random error term called the unique error or uniqueness. In factor analysis, the unique errors associated with each variable are usually assumed to be independent of the factors. Additionally, in the common factor model, the unique errors are assumed to be mutually independent. The factor analysis model is expressed in the following equation:

\[ x - \mu = \Lambda f + e \]

where \( x \) is the \( p \) vector of observed values, \( \mu \) is the \( p \) vector of variable means, \( \Lambda \) is the \( p \times k \) matrix of factor loadings, \( f \) is the \( k \) vector of hypothesized underlying random factors, \( e \) is the \( p \) vector of hypothesized unique random errors, \( p \) is the number of variables in the observed variables, and \( k \) is the number of factors.

Because much of the computation in factor analysis was originally done by hand or was expensive on early computers, quick (but dirty) algorithms that made the calculations possible were developed. One result is the many factor extraction methods available today. Generally speaking, in the exploratory or model building phase of a factor analysis, a method of factor extraction that is not computationally intensive (such as principal components, principal factor, or image analysis) is used. If desired, a computationally intensive method is then used to obtain the final factors.

**Discriminant Analysis**

The class `DiscriminantAnalysis` allows linear or quadratic discrimination and the use of either reclassification, split sample, or the leaving-out-one methods in order to evaluate the rule. Moreover, `DiscriminantAnalysis` can be executed in an online mode, that is, one or more observations can be added to the rule during each invocation of `DiscriminantAnalysis`.

The mean vectors for each group of observations and an estimate of the common covariance matrix for all groups are input to `DiscriminantAnalysis`. Output from `DiscriminantAnalysis` are linear combinations of the observations, which at most separate the groups. These linear combinations may subsequently be used for discriminating between the groups. Their use in graphically displaying differences between the groups is possibly more important, however.
ClusterKMeans Class

public class Imsl.Stat.ClusterKMeans
Perform a K-means (centroid) cluster analysis.

ClusterKMeans is an implementation of Algorithm AS 136 by Hartigan and Wong (1979). It computes
K-means (centroid) Euclidean metric clusters for an input matrix starting with initial estimates of the K
cluster means. It allows for missing values (coded as NaN, \textit{not a number}) and for weights and
frequencies.

Let $p$ denote the number of variables to be used in computing the Euclidean distance between
observations. The idea in K-means cluster analysis is to find a clustering (or grouping) of the
observations so as to minimize the total within-cluster sums of squares. In this case, the total sums of
squares within each cluster is computed as the sum of the centered sum of squares over all nonmissing
values of each variable. That is,

$$
\phi = \sum_{i=1}^{K} \sum_{j=1}^{p} \sum_{m=1}^{n_i} f_{vm} w_{vm} \delta_{vm,j} (x_{vm,j} - \bar{x}_{ij})^2
$$

where $v_{im}$ denotes the row index of the $m$-th observation in the $i$-th cluster in the matrix $X$; $n_i$ is the
number of rows of $X$ assigned to group $i$; $f$ denotes the frequency of the observation; $w$ denotes its
weight; $d$ is zero if the $j$-th variable on observation $v_{im}$ is missing, otherwise $\delta$ is one; and $\bar{x}_{ij}$ is the
average of the nonmissing observations for variable $j$ in group $i$. This method sequentially processes
each observation and reassigns it to another cluster if doing so results in a decrease in the total
within-cluster sums of squares. See Hartigan and Wong (1979) or Hartigan (1975) for details.

Property

MaxIterations
public int MaxIterations {get; set; }

Description
The maximum number of iterations.

Property Value
A int scalar specifying the maximum number of iterations.

Remarks
By default, MaxIterations = 30.
Constructor

ClusterKMeans
public ClusterKMeans(double[,] x, double[,] cs)

Description
Constructor for ClusterKMeans.

Parameters
  x – A double matrix containing the observations to be clustered.
  cs – A double matrix containing the cluster seeds, i.e. estimates for the cluster centers.

Exception
System.ArgumentException is thrown if x.GetLength(0), x.GetLength(1) are equal 0, or cs.GetLength(0) is less than 1

Methods

Compute
public double[,] Compute()

Description
Computes the cluster means.

Returns
A double matrix containing computed result.

Exceptions
  Imsl.Stat.NoConvergenceException is thrown if convergence did not occur within the maximum number of iterations
  Imsl.Stat.ClusterNoPointsException is thrown if the cluster seed yields a cluster with no points

GetClusterCounts
public int[] GetClusterCounts()

Description
Returns the number of observations in each cluster. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns
An int array containing the number of observations in each cluster.

GetClusterMembership
public int[] GetClusterMembership()
Returns the cluster membership for each observation. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns
An int array containing the cluster membership for each observation.

Remarks
Cluster membership 1 indicates the observation belongs to cluster 1, cluster membership 2 indicates the observation belongs to cluster 2, etc.

GetClusterSSQ
public double[] GetClusterSSQ()

Description
Returns the within sum of squares for each cluster. Note that the Compute method must be invoked first before invoking this method. Otherwise, the method throws a NullReferenceException exception.

Returns
A double array containing the within sum of squares for each cluster.

SetFrequencies
public void SetFrequencies(double[] frequencies)

Description
The frequency for each observation.

Parameter
 frequencies – A double array of size x.GetLength(0) containing the frequency for each observation.

Remarks
By default, Frequencies[] = 1.

SetWeights
public void SetWeights(double[] weights)

Description
Sets the weight for each observation.

Parameter
 weights – A double array of size x.GetLength(0) containing the weight for each observation.

Remarks
By default, Weights[] = 1.
Example: K-means Cluster Analysis

This example performs K-means cluster analysis on Fisher's iris data. The initial cluster seed for each iris type is an observation known to be in the iris type.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;

class ClusterKMeansEx1
{
    public static void Main(String[] argv)
    {
        double[,] x = {
            {5.100, 3.500, 1.400, 0.200},
            {4.900, 3.000, 1.400, 0.200},
            {4.700, 3.200, 1.300, 0.200},
            {4.600, 3.100, 1.500, 0.200},
            {5.000, 3.600, 1.400, 0.200},
            {5.400, 3.900, 1.700, 0.400},
            {4.600, 3.400, 1.400, 0.300},
            {5.000, 3.400, 1.500, 0.200},
            {5.000, 3.400, 1.500, 0.200},
            {5.400, 3.900, 1.700, 0.400},
            {4.600, 3.400, 1.400, 0.300},
            {5.000, 3.400, 1.500, 0.200},
            {4.400, 2.900, 1.400, 0.200},
            {4.900, 3.100, 1.500, 0.100},
            {5.400, 3.700, 1.500, 0.200},
            {4.800, 3.400, 1.600, 0.200},
            {4.800, 3.000, 1.400, 0.100},
            {4.300, 3.000, 1.100, 0.100},
            {5.800, 4.000, 1.200, 0.200},
            {5.700, 4.400, 1.500, 0.400},
            {5.400, 3.900, 1.300, 0.400},
            {5.100, 3.500, 1.400, 0.300},
            {5.700, 3.800, 1.700, 0.300},
            {5.100, 3.800, 1.500, 0.300},
            {5.400, 3.400, 1.700, 0.200},
            {5.100, 3.700, 1.500, 0.400},
            {4.600, 3.600, 1.000, 0.200},
            {5.100, 3.300, 1.700, 0.500},
        };
    }
}
```
Multivariate Analysis  
ClusterKMeans • 997

\{4.800, 3.400, 1.900, 0.200\},
\{5.000, 3.000, 1.600, 0.200\},
\{5.000, 3.400, 1.600, 0.400\},
\{5.200, 3.500, 1.500, 0.200\},
\{5.200, 3.400, 1.400, 0.200\},
\{4.700, 3.200, 1.600, 0.200\},
\{4.800, 3.100, 1.600, 0.200\},
\{5.400, 3.400, 1.500, 0.400\},
\{5.200, 4.100, 1.500, 0.100\},
\{5.500, 4.200, 1.400, 0.200\},
\{4.900, 3.100, 1.500, 0.200\},
\{5.000, 3.200, 1.200, 0.200\},
\{5.500, 3.500, 1.300, 0.200\},
\{4.900, 3.600, 1.400, 0.100\},
\{4.400, 3.000, 1.300, 0.200\},
\{5.100, 3.400, 1.500, 0.200\},
\{5.000, 3.500, 1.300, 0.300\},
\{4.500, 2.300, 1.300, 0.300\},
\{4.400, 3.200, 1.300, 0.200\},
\{5.000, 3.500, 1.600, 0.600\},
\{5.100, 3.800, 1.900, 0.400\},
\{4.800, 3.000, 1.400, 0.300\},
\{5.100, 3.800, 1.600, 0.200\},
\{4.600, 3.200, 1.400, 0.200\},
\{5.300, 3.700, 1.500, 0.200\},
\{5.000, 3.300, 1.400, 0.200\},
\{7.000, 3.200, 4.700, 1.400\},
\{6.400, 3.200, 4.500, 1.500\},
\{6.900, 3.100, 4.900, 1.500\},
\{5.500, 2.300, 4.000, 1.300\},
\{6.500, 2.800, 4.600, 1.500\},
\{5.700, 2.800, 4.500, 1.300\},
\{6.300, 3.300, 4.700, 1.600\},
\{4.900, 2.400, 3.300, 1.000\},
\{6.600, 2.900, 4.600, 1.300\},
\{5.200, 2.700, 3.900, 1.400\},
\{5.000, 2.000, 3.500, 1.000\},
\{5.900, 3.000, 4.200, 1.500\},
\{6.000, 2.200, 4.000, 1.000\},
\{6.100, 2.900, 4.700, 1.400\},
\{5.600, 2.900, 3.600, 1.300\},
\{6.700, 3.100, 4.400, 1.400\},
\{5.600, 3.000, 4.500, 1.500\},
\{5.800, 2.700, 4.100, 1.000\},
\{6.200, 2.200, 4.500, 1.500\},
\{5.600, 2.500, 3.900, 1.100\},
\{5.900, 3.200, 4.800, 1.800\},
\{6.100, 2.800, 4.000, 1.300\},
\{6.300, 2.500, 4.900, 1.500\},
\{6.100, 2.800, 4.700, 1.200\},
\{6.400, 2.900, 4.300, 1.300\},
\{6.600, 3.000, 4.400, 1.400\},
\{6.800, 2.800, 4.800, 1.400\},
\{6.700, 3.000, 5.000, 1.700\},
\{6.000, 2.900, 4.500, 1.500\},
\{5.700, 2.600, 3.500, 1.000\},
\{5.500, 2.400, 3.800, 1.100\},
\{5.500, 2.400, 3.700, 1.000\},
\{5.800, 2.700, 3.900, 1.200\},
\{6.000, 2.700, 5.100, 1.600\},
\{5.400, 3.000, 4.500, 1.500\},
\{6.000, 3.400, 4.500, 1.600\},
\{6.700, 3.100, 4.700, 1.500\},
\{6.300, 2.300, 4.400, 1.300\},
\{5.600, 3.000, 4.100, 1.300\},
\{5.500, 2.500, 4.000, 1.300\},
\{5.500, 2.600, 4.400, 1.200\},
\{6.100, 3.000, 4.600, 1.400\},
\{5.800, 2.600, 4.000, 1.200\},
\{5.000, 2.300, 3.300, 1.000\},
\{5.600, 2.700, 4.200, 1.300\},
\{5.700, 3.000, 4.200, 1.200\},
\{5.700, 2.900, 4.200, 1.300\},
\{6.200, 2.900, 4.300, 1.300\},
\{5.100, 2.500, 3.000, 1.100\},
\{5.700, 2.800, 4.100, 1.300\},
\{6.300, 3.300, 6.000, 2.500\},
\{5.800, 2.700, 5.100, 1.900\},
\{7.100, 3.000, 5.900, 2.100\},
\{6.300, 2.900, 5.600, 1.800\},
\{6.500, 3.000, 5.800, 2.200\},
\{7.600, 3.000, 6.600, 2.100\},
\{4.900, 2.500, 4.500, 1.700\},
\{7.300, 2.900, 6.300, 1.800\},
\{6.700, 2.500, 5.800, 1.800\},
\{7.200, 3.600, 6.100, 2.500\},
\{6.500, 3.200, 5.100, 2.000\},
\{6.400, 2.700, 5.300, 1.900\},
\{6.800, 3.000, 5.500, 2.100\},
\{5.700, 2.500, 5.000, 2.000\},
\{5.800, 2.800, 5.100, 2.400\},
\{6.400, 3.200, 5.300, 2.300\},
\{6.500, 3.000, 5.500, 1.800\},
\{7.700, 3.800, 6.700, 2.200\},
\{7.700, 2.600, 6.900, 2.300\},
\{6.000, 2.200, 5.000, 1.500\},
\{6.900, 3.200, 5.700, 2.300\},
\{5.600, 2.800, 4.900, 2.000\},
\{7.700, 2.800, 6.700, 2.000\},
\{6.300, 2.700, 4.900, 1.800\},
\{6.700, 3.300, 5.700, 2.100\},
\{7.200, 3.200, 6.000, 1.800\},
\{6.200, 2.800, 4.800, 1.800\},
\{6.100, 3.000, 4.900, 1.800\},
\{6.400, 2.800, 5.600, 2.100\},
\{7.200, 3.000, 5.800, 1.600\},
\{7.400, 2.800, 6.100, 1.900\},
\{7.900, 3.800, 6.400, 2.000\},
\{6.400, 2.800, 5.600, 2.200\},
\{6.300, 2.800, 5.100, 1.500\},
\{6.100, 2.600, 5.600, 1.400\},
\{7.700, 3.000, 6.100, 2.300\},
Output

Cluster Means

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.0060</td>
<td>3.4280</td>
<td>1.4620</td>
<td>0.2460</td>
</tr>
<tr>
<td>1</td>
<td>5.9016</td>
<td>2.7484</td>
<td>4.3935</td>
<td>1.4339</td>
</tr>
<tr>
<td>2</td>
<td>6.8500</td>
<td>3.0737</td>
<td>5.7421</td>
<td>2.0711</td>
</tr>
</tbody>
</table>

Cluster Membership

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5 1
6 1
7 1
8 1
9 1
10 1
11 1
12 1
13 1
14 1
15 1
16 1
17 1
18 1
19 1
20 1
21 1
22 1
23 1
24 1
25 1
26 1
27 1
28 1
29 1
30 1
31 1
32 1
33 1
34 1
35 1
36 1
37 1
38 1
39 1
40 1
41 1
42 1
43 1
44 1
45 1
46 1
47 1
48 1
49 1
50 2
51 2
52 3
53 2
54 2
55 2
56 2
57 2
58 2
59 2
60 2
Multivariate Analysis

ClusterKMeans • 1001
Dissimilarities Class

public class Imsl.Stat.Dissimilarities
Computes a matrix of dissimilarities (or similarities) between the columns (or rows) of a matrix.

Class Dissimilarities computes an upper triangular matrix (excluding the diagonal) of dissimilarities (or similarities) between the columns (or rows) of a matrix. Nine different distance measures can be computed. For the first three measures, three different scaling options can be employed. The distance matrix computed is generally used as input to clustering or multidimensional scaling functions.

The following discussion assumes that the distance measure is being computed between the columns of the matrix. If distances between the rows of the matrix are desired, use Row = true.

The distance method and scaling option used by Dissimilarities can be set via properties DistanceMethod and ScalingOption, respectively. For distance methods L2Norm, L1Norm, or InfinityNorm, each row of x is first scaled according to the value of ScalingOption. The scaling parameters are obtained from the values in the row scaled as either the standard deviation of the row or the row range; the standard deviation is computed from the unbiased estimate of the variance. If no scaling is performed, the parameters in the following discussion are all 1.0 (see ScalingOption). Once the scaling value (if any) has been computed, the distance between column i and column j is computed via the difference vector \( z_k = (x_k - y_k), i = 1, \ldots, ndstm \), where \( x_k \) denotes the k-th element in the i-th column, \( y_k \) denotes the corresponding element in the j-th column, and ndstm is the number of rows if differencing columns and the number of columns if differencing rows. For given \( z_i \), the distance methods that allow scaling are defined as:

<table>
<thead>
<tr>
<th>DistanceMethod</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2Norm</td>
<td>Euclidean distance (L_2 norm)</td>
</tr>
<tr>
<td>L1Norm</td>
<td>Sum of the absolute differences (L_1 norm)</td>
</tr>
<tr>
<td>InfinityNorm</td>
<td>Maximum difference (L_{\infty} norm)</td>
</tr>
</tbody>
</table>

The following distance measures do not allow for scaling.

<table>
<thead>
<tr>
<th>DistanceMethod</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mahalanobis</td>
<td>Mahalanobis distance</td>
</tr>
<tr>
<td>AbsCosine</td>
<td>Absolute value of the cosine of the angle between the vectors</td>
</tr>
<tr>
<td>AngleInRadians</td>
<td>Angle in radians (0, \pi) between the lines through the origin defined by the vectors</td>
</tr>
<tr>
<td>CorrelationCoefficient</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>AbsCorrelationCoefficient</td>
<td>Absolute value of the correlation coefficient</td>
</tr>
<tr>
<td>ExactMatches</td>
<td>Number of exact matches, where ( x_i = y_i ).</td>
</tr>
</tbody>
</table>

For the Mahalanobis distance, any variable used in computing the distance measure that is (numerically) linearly dependent upon the previous variables in the Index property is omitted from the distance measure.
Properties

DistanceMatrix
virtual public double[,] DistanceMatrix {get; }

Description
The distance matrix.

DistanceMethod
virtual public Imsl.Stat.Dissimilarities.Method DistanceMethod {get; set; }

Description
The method in computing the dissimilarities or similarities.

Property Value
A Dissimilarities.Method identifying the method to use in computing the dissimilarities or similarities.

Remarks
Acceptable values of DistanceMethod are:

<table>
<thead>
<tr>
<th>DistanceMethod</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2Norm</td>
<td>Euclidean distance ($L_2$ norm)</td>
</tr>
<tr>
<td>L1Norm</td>
<td>Sum of the absolute differences ($L_1$ norm)</td>
</tr>
<tr>
<td>InfinityNorm</td>
<td>Maximum difference ($L_{\infty}$ norm)</td>
</tr>
<tr>
<td>Mahalanobis</td>
<td>Mahalanobis distance</td>
</tr>
<tr>
<td>AbsCosine</td>
<td>Absolute value of the cosine of the angle between the vectors</td>
</tr>
<tr>
<td>AngleInRadians</td>
<td>Angle in radians (0, $\pi$) between the lines through the origin defined by the vectors</td>
</tr>
<tr>
<td>CorrelationCoefficient</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>AbsCorrelationCoefficient</td>
<td>Absolute value of the correlation coefficient</td>
</tr>
<tr>
<td>ExactMatches</td>
<td>Number of exact matches, where $x_i = y_i$.</td>
</tr>
</tbody>
</table>

See class description for more details. By default, DistanceMethod = Method.L2Norm.

Index
virtual public int[] Index {get; set; }

Description
The indices of the rows.

Property Value
An int array containing the indices of the columns (rows if Row = false) to use in computing the distance measure.
Remarks
By default, if Row = true, Index = 0, 1, ..., x.GetLength(1)-1. if Row = false, Index = 0, 1, ..., x.GetLength(0)-1, see Row.

**NumberOfProcessors**
public int NumberOfProcessors {get; set; }

**Description**
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**
An int indicating the maximum possible number of processors to use.

**Remarks**
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

**Row**
virtual public bool Row {get; set; }

**Description**
Identifies whether distances are computed between rows or columns of x.

**Property Value**
A bool identifying whether distances are computed between rows or columns of x.

**Remarks**
If Row = true, distances are computed between the rows of x. Otherwise, distances between the columns of x are computed. By default, Row = true.

**ScalingOption**
virtual public Imsl.Stat.Dissimilarities.Scaling ScalingOption {get; set; }

**Description**
The scaling option used if the L2Norm, L1Norm, or InfinityNorm distance methods are specified. See DistanceMethod.

**Property Value**
A Dissimilarities.Scaling containing the scaling option.

**Remarks**
By default, ScalingOption = Scaling.None.
### ScalingOption

<table>
<thead>
<tr>
<th>ScalingOption</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No scaling is performed.</td>
</tr>
<tr>
<td>StdDev</td>
<td>If Row = false, scale each column by the standard deviation of the column. If Row = true, scale each row by the standard deviation of the row.</td>
</tr>
<tr>
<td>Range</td>
<td>If Row = false, scale each column by the range of the column. If Row = true, scale each row by the range of the row.</td>
</tr>
</tbody>
</table>

### Constructor

**Dissimilarities**

```csharp
public Dissimilarities(double[,] x)
```

**Description**

Constructor for **Dissimilarities**.

**Parameter**

- `x` – A double matrix containing the data input matrix.

### Method

**Compute**

```csharp
public void Compute()
```

**Description**

Computes a matrix of dissimilarities (or similarities) between the columns (or rows) of a matrix.

**Exceptions**

- `Imsl.Stat.ScaleFactorZeroException` is thrown when computations cannot continue because a scale factor is zero.
- `Imsl.Stat.NoPositiveVarianceException` is thrown when no variable has positive variance.
- `Imsl.Stat.ZeroNormException` is thrown when the Euclidean norm of a column is equal to zero.

### Example: Dissimilarities

The following example illustrates the use of Dissimilarities for computing the Euclidean distance between the rows of a matrix:

```csharp
using System;
using Imsl.Math;
```
using Imsl.Stat;

public class DissimilaritiesEx1
{
    public static void Main(String[] args)
    {
        double[,] x = { { 1.0, 1.0 }, { 1.0, 0.0 },
                         { 1.0, -1.0 }, { 1.0, 2.0 }};

        Dissimilarities dist = new Dissimilarities(x);
        dist.Compute();
        new PrintMatrix("Distance Matrix").Print(dist.DistanceMatrix);
    }
}

Output
Distance Matrix
0 1 2 3
0 0 1 2 1
1 0 0 1 2
2 0 0 0 3
3 0 0 0 0

---

Dissimilarities.Method Enumeration


Specifies the type of distance method.

Fields

AbsCorrelationCoefficient
public Imsl.Stat.Dissimilarities.Method AbsCorrelationCoefficient

Description
Indicates the absolute value of the correlation coefficient distance method.

AbsCosine

Description
Indicates the absolute value of the cosine of the angle between the vectors distance method.
AngleInRadians

Description
Indicates the angle in radians \((0, \pi)\) between the lines through the origin defined by the vectors distance method.

CorrelationCoefficient

Description
Indicates the correlation coefficient distance method.

ExactMatches
public Imsl.Stat.Dissimilarities.Method ExactMatches

Description
Indicates the number of exact matches distance method.

InfinityNorm

Description
Indicates the maximum difference \((L_\infty \text{ norm})\) distance method.

L1Norm

Description
Indicates the sum of the absolute differences \((L_1 \text{ norm})\) distance method.

L2Norm

Description
Indicates the Euclidean distance method \((L_2 \text{ norm})\).

Mahalanobis

Description
Indicates the Mahalanobis distance method.
Dissimilarities.Scaling Enumeration

Specifies the type of scaling.

Fields

None
public Imsl.Stat.Dissimilarities.Scaling None

Description
Indicates no scaling.

Range
public Imsl.Stat.Dissimilarities.Scaling Range

Description
Indicates scaling by the range.

StdDev

Description
Indicates scaling by the standard deviation.

ClusterHierarchical Class

public class Imsl.Stat.ClusterHierarchical
Performs a hierarchical cluster analysis from a distance matrix.

Class ClusterHierarchical conducts a hierarchical cluster analysis based upon a distance matrix, or, by appropriate use of the transformation specified in the TransformType property, based upon a similarity matrix. Only the upper triangular part of the input matrix is used.

Hierarchical clustering in ClusterHierarchical proceeds as follows:
Initially, each data point is considered to be a cluster, numbered 1 to n, where n is the number of rows in the input matrix, dist.

Multivariate Analysis  Dissimilarities.Scaling • 1009
1. If the input matrix contains similarities, the matrix is transformed to a distance matrix using the transform type specified by the property TransformType. Set \( k = 1 \).

2. A search is made of the distance matrix to find the two closest clusters. These clusters are merged to form a new cluster, numbered \( n + k \). The cluster numbers of the two clusters joined at this stage are saved as Right Sons and Left Sons, and the distance measure between the two clusters is stored as Cluster Level.

3. Based upon the method of clustering, updating of the distance measure in the row and column of dist corresponding to the new cluster is performed.

4. Set \( k = k + 1 \). If \( k \) is less than \( n \), go to Step 2.

The five methods differ primarily in how the distance matrix is updated after two clusters have been joined. The Method property specifies how the distance of the cluster just merged with each of the remaining clusters will be updated. Class ClusterHierarchical allows five methods for computing the distances. To understand these measures, suppose in the following discussion that clusters A and B have just been joined to form cluster Z, and interest is in computing the distance of Z with another cluster called C.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Single linkage (minimum distance). The distance from $Z$ to $C$ is the minimum of the distances ($A$ to $C$, $B$ to $C$).</td>
</tr>
<tr>
<td>Complete</td>
<td>Complete linkage (maximum distance). The distance from $Z$ to $C$ is the maximum of the distances ($A$ to $C$, $B$ to $C$).</td>
</tr>
<tr>
<td>AvgWithinClusters</td>
<td>Average-distance-within-clusters method. The distance from $Z$ to $C$ is the average distance of all objects that would be within the cluster formed by merging clusters $Z$ and $C$. This average may be computed according to formulas given by Anderberg (1973, page 139).</td>
</tr>
<tr>
<td>AvgBetweenClusters</td>
<td>Average-distance-between-clusters method. The distance from $Z$ to $C$ is the average distance of objects within cluster $Z$ to objects within cluster $C$. This average may be computed according to methods given by Anderberg (1973, page 140).</td>
</tr>
<tr>
<td>Wards</td>
<td>Ward’s method: Clusters are formed so as to minimize the increase in the within-cluster sums of squares. The distance between two clusters is the increase in these sums of squares if the two clusters were merged. A method for computing this distance from a squared Euclidean distance matrix is given by Anderberg (1973, pages 142-145).</td>
</tr>
</tbody>
</table>

In general, single linkage will yield long thin clusters while complete linkage will yield clusters that are more spherical. Average linkage and Ward’s linkage tend to yield clusters that are similar to those obtained with complete linkage.

Class `ClusterHierarchical` produces a unique representation of the binary cluster tree via the following three conventions; the fact that the tree is unique should aid in interpreting the clusters. First, when two clusters are joined and each cluster contains two or more data points, the cluster initially formed with the smallest level becomes the left son. Second, when a cluster containing more than one data point is joined with a cluster containing a single data point, the cluster with the single data point becomes the right son. Third, when two clusters containing only one object are joined, the cluster with the smallest cluster number becomes the right son.

**Comments**

1. The clusters corresponding to the original data points are numbered from 1 to $n$, where $n$ is the number of rows in `dist`. The $n-1$ clusters formed by merging clusters are numbered $n+1$ to $n+(n-1)$.

2. Raw correlations, if used as similarities, should be made positive and transformed to a distance measure. One such transformation can be performed by setting `TransformType = ReciprocalAbs`.

3. The user may cluster either variables or observations with `ClusterHierarchical` since a dissimilarity matrix, not the original data, is used. Class `Imsl.Stat.Dissimilarities` (p. 1002) may be used to compute the matrix `dist` for either the variables or observations.
Properties

**ClusterLeftSons**

```csharp
virtual public int[] ClusterLeftSons {get; }
```

**Description**
The left sons of each merged cluster.

**Property Value**
An int array containing the left sons of each merged cluster.

**ClusterLevel**

```csharp
virtual public double[] ClusterLevel {get; }
```

**Description**
The level at which the clusters are joined.

**Property Value**
A double array containing the level at which the clusters are joined.

**Remarks**
Element $[k-1]$ contains the distance (or similarity) level at which cluster $n + k$ was formed. If the original data in dist was transformed, the inverse transformation is applied to the returned values.

**ClusterRightSons**

```csharp
virtual public int[] ClusterRightSons {get; }
```

**Description**
The right sons of each merged cluster.

**Property Value**
An int array containing the right sons of each merged cluster.

**Method**

```csharp
virtual public Imsl.Stat.ClusterHierarchical.Linkage Method {get; set; }
```

**Description**
The clustering method.

**Property Value**
A ClusterHierarchical.Linkage identifying the clustering method to use.

**Remarks**
By Default, Method = Linkage.Single.
### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Single linkage (minimum distance).</td>
</tr>
<tr>
<td>Complete</td>
<td>Complete linkage (maximum distance).</td>
</tr>
<tr>
<td>AvgWithinClusters</td>
<td>Average distance within (average distance between objects within the merged cluster).</td>
</tr>
<tr>
<td>AvgBetweenClusters</td>
<td>Average distance between (average distance between objects in the two clusters).</td>
</tr>
<tr>
<td>Wards</td>
<td>Ward’s method (minimize the within-cluster sums of squares). For Ward’s method, the elements of dist are assumed to be Euclidean distances.</td>
</tr>
</tbody>
</table>

### NumberOfProcessors

```csharp
public int NumberOfProcessors {get; set; }
```

**Description**

Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

**Property Value**

An int indicating the maximum possible number of processors to use.

**Remarks**

By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

### TransformType

```csharp
virtual public Imsl.Stat.ClusterHierarchical.Transformation TransformType {get; set; }
```

**Description**

The type of transformation.

**Property Value**

A ClusterHierarchical.Transformation identifying the type of transformation applied to the measures in dist.

**Remarks**

By Default, TransformType = Transformation.None.

<table>
<thead>
<tr>
<th>TransformType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No transformation is required. The elements of dist are distances.</td>
</tr>
<tr>
<td>Multiplication</td>
<td>Convert similarities to distances by multiplying by -1.0.</td>
</tr>
<tr>
<td>ReciprocalAbs</td>
<td>Convert similarities (usually correlations) to distances by taking the reciprocal of the absolute value.</td>
</tr>
</tbody>
</table>
Constructor

ClusterHierarchical
public ClusterHierarchical(double[, ] dist)

Description
Constructor for ClusterHierarchical.

Parameter
dist – A double symmetric matrix containing the distance (or similarity) matrix.

Remarks
Only the upper triangular part of dist needs to be present.

Methods

Compute
public void Compute()

Description
Performs a hierarchical cluster analysis.

GetClusterMembership
public int[] GetClusterMembership(int nClusters)

Description
Returns the cluster membership of each observation.

Parameter
nClusters – An int which specifies the desired number of clusters.

Returns
An int array containing the cluster membership of each observation.

GetObsPerCluster
public int[] GetObsPerCluster(int nClusters)

Description
Returns the number of observations in each cluster.

Parameter
nClusters – An int which specifies the desired number of clusters.

Returns
An int array containing the number of observations in each cluster.
Example: ClusterHierarchical

This example illustrates a typical usage of ClusterHierarchical. The Fisher iris data is clustered. First the distance between irises is computed using the class Dissimilarities. The resulting distance matrix is then clustered using ClusterHierarchical, and cluster memberships for 5 clusters are computed.

```csharp
using System;
using Imsl.Stat;

public class ClusterHierarchicalEx1
{
    public static void Main(String[] args)
    {
        double[,] irisData = {
            { 5.1, 3.5, 1.4, .2},
            { 4.9, 3.0, 1.4, .2},
            { 4.7, 3.2, 1.3, .2},
            { 4.6, 3.1, 1.5, .2},
            { 5.0, 3.6, 1.4, .2},
            { 5.4, 3.9, 1.7, .4},
            { 4.6, 3.4, 1.4, .3},
            { 5.0, 3.4, 1.5, .2},
            { 4.4, 2.9, 1.4, .2},
            { 4.9, 3.1, 1.5, .1},
            { 5.4, 3.7, 1.5, .2},
            { 4.8, 3.4, 1.6, .2},
            { 4.8, 3.0, 1.4, .1},
            { 4.3, 3.0, 1.1, .1},
            { 5.8, 4.0, 1.2, .2},
            { 5.7, 4.4, 1.5, .4},
            { 5.4, 3.9, 1.3, .4},
            { 5.1, 3.5, 1.4, .3},
            { 5.7, 3.8, 1.7, .3},
            { 5.4, 3.4, 1.7, .2},
            { 5.1, 3.7, 1.5, .4},
            { 4.6, 3.6, 1.0, .2},
            { 5.1, 3.3, 1.7, .5},
            { 4.8, 3.4, 1.9, .2},
            { 5.0, 3.0, 1.6, .2},
            { 5.0, 3.4, 1.6, .4},
            { 5.2, 3.5, 1.5, .2},
            { 5.2, 3.4, 1.4, .2},
            { 4.7, 3.2, 1.6, .2},
            { 4.8, 3.1, 1.6, .2},
            { 5.4, 3.4, 1.5, .4},
            { 5.2, 4.1, 1.5, .1},
            { 5.5, 4.2, 1.4, .2},
            { 4.9, 3.1, 1.5, .2},
            { 5.0, 3.2, 1.2, .2},
            { 5.5, 3.5, 1.3, .2},
            { 4.9, 3.6, 1.4, .1},
            { 4.4, 3.0, 1.3, .2},
            { 5.1, 3.4, 1.5, .2},
        };
{ 5.0, 3.5, 1.3, .3},
{ 4.5, 2.3, 1.3, .3},
{ 4.4, 3.2, 1.3, .2},
{ 5.0, 3.5, 1.6, .6},
{ 5.1, 3.8, 1.9, .4},
{ 4.8, 3.0, 1.4, .3},
{ 5.1, 3.8, 1.6, .2},
{ 4.6, 3.2, 1.4, .2},
{ 5.3, 3.7, 1.5, .2},
{ 5.0, 3.3, 1.4, .2},
{ 7.0, 3.2, 4.7, 1.4},
{ 6.4, 3.2, 4.5, 1.5},
{ 6.9, 3.1, 4.9, 1.5},
{ 5.5, 2.3, 4.0, 1.3},
{ 6.5, 2.8, 4.6, 1.5},
{ 5.7, 2.8, 4.5, 1.3},
{ 6.3, 3.3, 4.7, 1.6},
{ 4.9, 2.4, 3.3, 1.0},
{ 6.6, 2.9, 4.6, 1.3},
{ 5.2, 2.7, 3.9, 1.4},
{ 5.0, 2.0, 3.5, 1.0},
{ 5.9, 3.0, 4.2, 1.5},
{ 6.0, 2.2, 4.0, 1.0},
{ 6.1, 2.9, 4.7, 1.4},
{ 5.6, 2.9, 3.6, 1.3},
{ 6.7, 3.1, 4.4, 1.4},
{ 5.6, 3.0, 4.5, 1.5},
{ 5.8, 2.7, 4.1, 1.0},
{ 6.2, 2.2, 4.5, 1.5},
{ 5.6, 2.5, 3.9, 1.1},
{ 5.9, 3.2, 4.8, 1.8},
{ 6.1, 2.8, 4.0, 1.3},
{ 6.3, 2.5, 4.9, 1.5},
{ 6.1, 2.8, 4.7, 1.2},
{ 6.4, 2.9, 4.3, 1.3},
{ 6.6, 3.0, 4.4, 1.4},
{ 6.8, 2.8, 4.8, 1.4},
{ 6.7, 3.0, 5.0, 1.7},
{ 6.0, 2.9, 4.5, 1.5},
{ 5.7, 2.6, 3.5, 1.0},
{ 5.5, 2.4, 3.8, 1.1},
{ 5.5, 2.4, 3.7, 1.0},
{ 5.8, 2.7, 3.9, 1.2},
{ 6.0, 2.7, 5.1, 1.6},
{ 5.4, 3.0, 4.5, 1.5},
{ 6.0, 3.4, 4.5, 1.6},
{ 6.7, 3.1, 4.7, 1.5},
{ 6.3, 2.3, 4.4, 1.3},
{ 5.6, 3.0, 4.1, 1.3},
{ 5.5, 2.5, 4.0, 1.3},
{ 5.5, 2.6, 4.4, 1.2},
{ 6.1, 3.0, 4.6, 1.4},
{ 5.8, 2.6, 4.0, 1.2},
{ 5.0, 2.3, 3.3, 1.0},
{ 5.6, 2.7, 4.2, 1.3},
{ 5.7, 3.0, 4.2, 1.2},
{ 5.7, 2.9, 4.2, 1.3},
{ 6.2, 2.9, 4.3, 1.3},
{ 5.1, 2.5, 3.0, 1.1},
{ 5.7, 2.8, 4.1, 1.3},
{ 6.3, 3.3, 6.0, 2.5},
{ 5.8, 2.7, 5.1, 1.9},
{ 7.1, 3.0, 5.9, 2.1},
{ 6.3, 2.9, 5.6, 1.8},
{ 6.5, 3.0, 5.8, 2.2},
{ 7.6, 3.0, 6.6, 2.1},
{ 4.9, 2.5, 4.5, 1.7},
{ 7.3, 2.9, 6.3, 1.8},
{ 6.7, 2.5, 5.8, 1.8},
{ 7.2, 3.6, 6.1, 2.5},
{ 6.5, 3.2, 5.1, 2.0},
{ 6.4, 2.7, 5.3, 1.9},
{ 6.8, 3.0, 5.5, 2.1},
{ 5.7, 2.5, 5.0, 2.0},
{ 5.8, 2.8, 5.1, 2.4},
{ 6.4, 3.2, 5.3, 2.3},
{ 6.5, 3.0, 5.5, 1.8},
{ 7.7, 3.8, 6.7, 2.2},
{ 7.7, 2.6, 6.9, 2.3},
{ 6.0, 2.2, 5.0, 1.5},
{ 6.9, 3.2, 5.7, 2.3},
{ 5.6, 2.8, 4.9, 2.0},
{ 7.7, 2.8, 6.7, 2.0},
{ 6.3, 2.7, 4.9, 1.8},
{ 6.7, 3.3, 5.7, 2.1},
{ 7.2, 3.2, 6.0, 1.8},
{ 6.2, 2.8, 4.8, 1.8},
{ 6.1, 3.0, 4.9, 1.8},
{ 6.4, 2.8, 5.6, 2.1},
{ 7.2, 3.0, 5.8, 1.6},
{ 7.4, 2.8, 6.1, 1.9},
{ 7.9, 3.8, 6.4, 2.0},
{ 6.4, 2.8, 5.6, 2.2},
{ 6.3, 2.8, 5.1, 1.5},
{ 6.1, 2.6, 5.6, 1.4},
{ 7.7, 3.0, 6.1, 2.3},
{ 6.3, 3.4, 5.6, 2.4},
{ 6.4, 3.1, 5.5, 1.8},
{ 6.0, 3.0, 4.8, 1.8},
{ 6.9, 3.1, 5.4, 2.1},
{ 6.7, 3.1, 5.6, 2.4},
{ 6.9, 3.1, 5.1, 2.3},
{ 5.8, 2.7, 5.1, 1.9},
{ 6.8, 3.2, 5.9, 2.3},
{ 6.7, 3.3, 5.7, 2.3},
{ 6.7, 3.0, 5.2, 2.3},
{ 6.3, 2.5, 5.0, 1.9},
{ 6.5, 3.0, 5.2, 2.0},
{ 6.2, 3.4, 5.4, 2.3},
{ 5.9, 3.0, 5.1, 1.8});

Dissimilarities dist = new Dissimilarities(irisData);
dist.ScalingOption = Dissimilarities.Scaling.StdDev;
dist.Compute();
ClusterHierarchical clink = new ClusterHierarchical(dist.DistanceMatrix);
clink.Method = ClusterHierarchical.Linkage.AvgWithinClusters;
clink.Compute();

int nClusters = 5;
int[] iclus = clink.GetClusterMembership(nClusters);

Output
Cluster Membership
5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
2 3 2 3 2 1 4 1 3 2
2 3 2 3 2 3 2 1 4
1 2 3 3 1 2 3 3 2
2 2 3 2 2 3 3 3

Observations Per Cluster
8 19 44 29 50

ClusterHierarchical.Linkage Enumeration

Specifies the type of linkage.

**Fields**

**AvgBetweenClusters**

public class Imsl.Stat.ClusterHierarchical.Linkage AvgBetweenClusters

**Description**

Indicates the average distance between (average distance between objects in the two clusters) method.

**AvgWithinClusters**

public class Imsl.Stat.ClusterHierarchical.Linkage AvgWithinClusters

**Description**

Indicates the average distance within (average distance between objects within the merged cluster) method.

**Complete**


**Description**

Indicates the complete linkage (maximum distance) method.

**Single**


**Description**

Indicates the single linkage (minimum distance) method.

**Wards**

public class Imsl.Stat.ClusterHierarchical.Linkage Wards

**Description**

Indicates the Ward’s method.

**ClusterHierarchical.Transformation Enumeration**


Specifies the type of transformation.
### Fields

**Multiplication**

```java
```

**Description**

Indicates transformation by multiplication by -1.0.

**None**

```java
```

**Description**

Indicates no transformation.

**ReciprocalAbs**

```java
```

**Description**

Indicates transformation by taking the reciprocal of the absolute value.

### FactorAnalysis Class

```java
public class Imsl.Stat.FactorAnalysis
```

Performs Principal Component Analysis or Factor Analysis on a covariance or correlation matrix.

Class `FactorAnalysis` computes principal components or initial factor loading estimates for a variance-covariance or correlation matrix using exploratory factor analysis models.

Models available are the principal component model for factor analysis and the common factor model with additions to the common factor model in alpha factor analysis and image analysis. Methods of estimation include principal components, principal factor, image analysis, unweighted least squares, generalized least squares, and maximum likelihood.

For the principal component model there are methods to compute the characteristic roots, characteristic vectors, standard errors for the characteristic roots, and the correlations of the principal component scores with the original variables. Principal components obtained from correlation matrices are the same as principal components obtained from standardized (to unit variance) variables.

The principal component scores are the elements of the vector \( y = \Gamma^T x \) where \( \Gamma \) is the matrix whose columns are the characteristic vectors (eigenvectors) of the sample covariance (or correlation) matrix and \( x \) is the vector of observed (or standardized) random variables. The variances of the principal component scores are the characteristic roots (eigenvalues) of the covariance (correlation) matrix.
Asymptotic variances for the characteristic roots were first obtained by Girshick (1939) and are given more recently by Kendall, Stuart, and Ord (1983, page 331). These variances are computed either for variance-covariance matrices or for correlation matrices.

The correlations of the principal components with the observed (or standardized) variables are the same as the unrotated factor loadings obtained for the principal components model for factor analysis when a correlation matrix is input.

In the factor analysis model used for factor extraction, the basic model is given as $\Sigma = \Lambda \Lambda^T + \Psi$ where $\Sigma$ is the $p \times p$ population covariance matrix. $\Lambda$ is the $p \times k$ matrix of factor loadings relating the factors $f$ to the observed variables $x$, and $\Psi$ is the $p \times p$ matrix of covariances of the unique errors $e$. Here, $p$ represents the number of variables and $k$ is the number of factors. The relationship between the factors, the unique errors, and the observed variables is given as $x = \Lambda f + e$ where, in addition, it is assumed that the expected values of $e, f$, and $x$ are zero. (The sample means can be subtracted from $x$ if the expected value of $x$ is not zero.) It is also assumed that each factor has unit variance, the factors are independent of each other, and that the factors and the unique errors are mutually independent. In the common factor model, the elements of the vector of unique errors $e$ are also assumed to be independent of one another so that the matrix $\Psi$ is diagonal. This is not the case in the principal component model in which the errors may be correlated.

Further differences between the various methods concern the criterion that is optimized and the amount of computer effort required to obtain estimates. Generally speaking, the least-squares and maximum likelihood methods, which use iterative algorithms, require the most computer time with the principal factor, principal component, and the image methods requiring much less time since the algorithms in these methods are not iterative. The algorithm in alpha factor analysis is also iterative, but the estimates in this method generally require somewhat less computer effort than the least-squares and maximum likelihood estimates. In all algorithms one eigensystem analysis is required on each iteration.

### Properties

<table>
<thead>
<tr>
<th>ConvergenceCriterion1</th>
</tr>
</thead>
</table>

```csharp
public double ConvergenceCriterion1 {get; set; }
```

**Description**
The convergence criterion used to terminate the iterations.

**Property Value**
A double used to terminate the iterations.

**Remarks**
For the least squares and and maximum likelihood methods convergence is assumed when the relative change in the criterion is less than `ConvergenceCriterion1`. For alpha factor analysis, convergence is assumed when the maximum change (relative to the variance) of a uniqueness is less than `ConvergenceCriterion1`. `ConvergenceCriterion1` is not referenced for the other estimation methods. By default, `ConvergenceCriterion1` is set to 0.0001.

**Multivariate Analysis**

**FactorAnalysis** • 1021
ConvergenceCriterion2
public double ConvergenceCriterion2 {get; set; }

Description
The convergence criterion used to switch to exact second derivatives.

Property Value
A double used to switch to exact second derivatives.

Remarks
When the largest relative change in the unique standard deviation vector is less than ConvergenceCriterion2, exact second derivative vectors are used. By default, ConvergenceCriterion2 is set to 0.1. Not referenced for principal component, principal factor, image factor, or alpha factor methods.

DegreesOfFreedom
public int DegreesOfFreedom {get; set; }

Description
The number of degrees of freedom.

Property Value
An int value specifying the number of degrees of freedom in the input matrix.

Remarks
If this property is not set, 100 degrees of freedom are assumed.

FactorLoadingEstimationMethod
public Imsl.Stat.FactorAnalysis.Model FactorLoadingEstimationMethod {get; set; }

Description
The factor loading estimation method.

Property Value
Indicates the method to be applied for obtaining the factor loadings. Use FactorAnalysis.Model field PrincipalComponent, PrincipalFactor, UnweightedLeastSquares, GeneralizedLeastSquares, MaximumLikelihood, ImageFactorAnalysis, or AlphaFactorAnalysis for FactorLoadingEstimationMethod. By default, the PrincipalComponent is used.

Remarks
For the principal component and principal factor methods, the factor loading estimates are computed as

\[
\hat{\Gamma} \hat{\Delta}^{-1/2}
\]

where \( \hat{\Gamma} \) and the diagonal matrix \( \hat{\Delta} \) are the eigenvalues and eigenvectors of a matrix. In the principal component model, the eigensystem analysis is performed on the sample covariance (correlation) matrix \( S \) while in the principal factor model the matrix \((S - \Psi)\) is used. If the unique error variances \( \Psi \) are not
known in the principal factor model, then they are estimated. This is achieved by setting the property `VarianceEstimationMethod` to 0. If the principal component model is used, the error variances in the `VarianceEstimationMethod` property are set to 0.0 automatically.

The basic idea in the principal component method is to find factors that maximize the variance in the original data that is explained by the factors. Because this method allows the unique errors to be correlated, some factor analysts insist that the principal component method is not a factor analytic method. Usually however, the estimates obtained via the principal component model and other models in factor analysis will be quite similar.

It should be noted that both the principal component and the principal factor methods give different results when the correlation matrix is used in place of the covariance matrix. Indeed, any rescaling of the sample covariance matrix can lead to different estimates with either of these methods. A further difficulty with the principal factor method is the problem of estimating the unique error variances. Theoretically, these must be known in advance and set using the the `VarianceEstimationMethod` property. In practice, the estimates of these parameters produced by setting the property `VarianceEstimationMethod` to 0 are often used. In either case, the resulting adjusted covariance (correlation) matrix

\[(S - \hat{\Psi})\]

may not yield the `nfactors` positive eigenvalues required for `nfactors` factors to be obtained. If this occurs, the user must either lower the number of factors to be estimated or give new unique error variance values.

For the least-squares and maximum likelihood methods an iterative algorithm is used to obtain the estimates (see Joreskog 1977). As with the principal factor model, the user may either input the initial unique error variances or allow the algorithm to compute initial estimates. Unlike the principal factor method, the code then optimizes the criterion function with respect to both \(\Psi\) and \(\Gamma\). (In the principal factor method, \(\Psi\) is assumed to be known. Given \(\Psi\), estimates for \(\Lambda\) may be obtained.)

The major differences between the estimation methods described in this member function are in the criterion function that is optimized. Let \(S\) denote the sample covariance (correlation) matrix, and let \(\Sigma\) denote the covariance matrix that is to be estimated by the factor model. In the unweighted least-squares method, also called the iterated principal factor method or the minres method (see Harman 1976, page 177), the function minimized is the sum of the squared differences between \(S\) and \(\Sigma\). This is written as

\[\Phi_{ul} = .5 \cdot \text{trace}((S - \Sigma)^2).\]

Generalized least-squares and maximum likelihood estimates are asymptotically equivalent methods. Maximum likelihood estimates maximize the (normal theory) likelihood

\[\{\Phi_{ml} = \text{trace}(\Sigma^{-1}S) - \log(|\Sigma^{-1}S|)\},\]

while generalized least squares optimizes the function

\[\Phi_{gs} = \text{trace}(\Sigma S^{-1} - I)^2.\]

In all three methods, a two-stage optimization procedure is used. This proceeds by first solving the likelihood equations for \(\Lambda\) in terms of \(\Psi\) and substituting the solution into the likelihood. This gives a criterion \(\Phi(\Psi, \Lambda(\Psi))\), which is optimized with respect to \(\Psi\). In the second stage, the estimates

\[\hat{\Psi}\]

are obtained from the estimates for \(\Psi\).

The generalized least-squares and the maximum likelihood methods allow for the computation of a statistic for testing that `nfactors` common factors are adequate to fit the model. This is a chi-squared
test that all remaining parameters associated with additional factors are zero. If the probability of a larger 
chi-squared is small (see \texttt{stat[4]}) so that the null hypothesis is rejected, then additional factors are 
needed (although these factors may not be of any practical importance). Failure to reject does not 
legitimize the model. The statistic \texttt{stat[2]} is a likelihood ratio statistic in maximum likelihood 
estimates. As such, it asymptotically follows a chi-squared distribution with degrees of freedom given in \texttt{stat[3]}.

The Tucker and Lewis (1973) reliability coefficient, \( \rho \), is returned in \texttt{stat[1]} when the maximum 
likelihood or generalized least-squares methods are used. This coefficient is an estimate of the ratio of 
explained to the total variation in the data. It is computed as follows:

\[
\rho = \frac{mM_o - mM_k}{mM_o - 1}
\]

\[
m = d - \frac{2p + 5}{6} - \frac{2k}{6}
\]

\[
M_o = \frac{-\ln(|S|)}{p(p - 1)/2}
\]

\[
M_k = \frac{\Phi}{((p - k)^2 - p - k)/2}
\]

where \(|S|\) is the determinant of \( \text{cov} \), \( p \) is the number of variables, \( k \) is the number of factors, \( \Phi \) is the 
optimized criterion, and \( d \) is the number of degrees of freedom.

The term “image analysis” is used here to denote the noniterative image method of Kaiser (1963). It is 
not the image factor analysis discussed by Harman (1976, page 226). The image method (as well as the 
alpha factor analysis method) begins with the notion that only a finite number from an infinite number of 
possible variables have been measured. The image factor pattern is calculated under the assumption that 
the ratio of the number of factors to the number of observed variables is near zero so that a very good 
estimate for the unique error variances (for standardized variables) is given as one minus the squared 
multiple correlation of the variable under consideration with all variables in the covariance matrix.

First, the matrix \( D^2 = (\text{diag}(S^{-1}))^{-1} \) is computed where the operator “\text{diag}” results in a matrix 
consisting of the diagonal elements of its argument, and \( S \) is the sample covariance (correlation) matrix. 
Then, the eigenvalues \( \Lambda \) and eigenvectors \( \Gamma \) of the matrix \( D^{-1}SD^{-1} \) are computed. Finally, the unrotated 
image factor pattern matrix is computed as \( A = D \Gamma [(\Lambda - I)^2 \Lambda^{-1}]^{1/2} \).

The alpha factor analysis method of Kaiser and Caffrey (1965) finds factor-loading estimates to 
maximize the correlation between the factors and the complete universe of variables of interest. The 
basic idea in this method is as follows: only a finite number of variables out of a much larger set of 
possible variables is observed. The population factors are linearly related to this larger set while the 
observed factors are linearly related to the observed variables. Let \( f \) denote the factors obtainable from a 
finite set of observed random variables, and let \( \xi \) denote the factors obtainable from the universe of 
observable variables. Then, the alpha method attempts to find factor-loading estimates so as to maximize 
the correlation between \( f \) and \( \xi \). In order to obtain these estimates, the iterative algorithm of Kaiser and 
Caffrey (1965) is used.

\[
\text{MaxIterations}
\]

\[
\text{public int MaxIterations } \{ \text{get; set; } \}
\]
**Description**
The maximum number of iterations in the iterative procedure.

**Property Value**
An int used as the maximum number of iterations allowed during the iterative portion of the algorithm.

**Remarks**
By default, **MaxIterations** is set to 60. **MaxIterations** is not referenced for factor loading methods **PrincipalComponent**, **PrincipalFactor**, or **ImageFactorAnalysis**.

---

### MaxStep

```csharp
public int MaxStep {get; set; }
```

**Description**
The maximum number of step halvings allowed during an iteration.

**Property Value**
An int used as the maximum number of step halvings allowed during an iteration.

**Remarks**
If this property is not set, **MaxStep** is set to 8. **MaxStep** is not referenced for **PrincipalComponent**, **PrincipalFactor**, **ImageFactorAnalysis**, or **AlphaFactorAnalysis** methods.

---

### NumberOfProcessors

```csharp
public int NumberOfProcessors {get; set; }
```

**Description**
Perform the parallel calculations with the maximum possible number of processors set to **NumberOfProcessors**.

**Property Value**
An int indicating the maximum possible number of processors to use.

**Remarks**
By default, **NumberOfProcessors** = **Environment.ProcessorCount**. If **NumberOfProcessors** is set to a number less than 1 or greater than **Environment.ProcessorCount**, **Environment.ProcessorCount** will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

---

### VarianceEstimationMethod

```csharp
public int VarianceEstimationMethod {get; set; }
```

**Description**
The variance estimation method.

**Property Value**
An int used to designate the method to be applied for obtaining the initial estimates of the unique variances.

---

**Multivariate Analysis**

**FactorAnalysis** • 1025
**Remarks**

By default, `VarianceEstimationMethod` is set to 1.

<table>
<thead>
<tr>
<th>init</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial estimates are taken as the constant (1-\text{nFactors}/(2\times\text{nVar})) divided by the diagonal elements of the inverse of input matrix <code>cov</code>.</td>
</tr>
<tr>
<td>1</td>
<td>Initial estimates are input by the user in vector <code>uniq</code>.</td>
</tr>
</tbody>
</table>

Note that when the factor loading estimation method is `PrincipalComponent`, the initial estimates in `uniq` are reset to 0.0.

**Constructor**

FactorAnalysis

```csharp
public FactorAnalysis(double[,] cov, Imsl.Stat.FactorAnalysis.MatrixType matrixType, int nfactors)
```

**Description**

Constructor for `FactorAnalysis`.

**Parameters**

- `cov` – A `double` matrix containing the covariance or correlation matrix.
- `matrixType` – An `int` scalar indicating the type of matrix that is input.
- `nfactors` – An `int` scalar indicating the number of factors in the model.

**Remarks**

`FactorAnalysis.matrixType` can specify a `VarianceCovariance` or `Correlation` matrix.

If `nfactors` is not known in advance, several different values of `nfactors` should be used, and the most reasonable value kept in the final solution. Since, in practice, the non- iterative methods often lead to solutions which differ little from the iterative methods, it is usually suggested that a non- iterative method be used in the initial tages of the factor analysis, and that the iterative methods be used once issues such as the number of factors have been resolved.

**Methods**

GetCorrelations

```csharp
public double[,] GetCorrelations()
```

**Description**

Returns the correlations of the principal components.
Returns
A double matrix containing the correlations of the principal components with the observed/standardized variables.

Remarks
If a covariance matrix is input to the constructor, then the correlations are with the observed variables. Otherwise, the correlations are with the standardized (to a variance of 1.0) variables. Only valid for the Principal Components Model.

Exceptions
- Imsl.Stat.RankException is thrown if the rank of the covariance matrix is less than the number of factors.
- Imsl.Stat.NotSemiDefiniteException is thrown if the Hessian matrix not semi-definite.
- Imsl.Stat.NotPositiveSemiDefiniteException is thrown if the covariance matrix is not positive semi-definite.
- Imsl.Stat.SingularException is thrown if the covariance matrix is singular.
- Imsl.Stat.BadVarianceException is thrown if the input variance is not in the allowed range.
- Imsl.Stat.EigenvalueException is thrown if an error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.
- Imsl.Stat.NonPositiveEigenvalueException is thrown if in alpha factor analysis an eigenvalue is not positive. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced, or new initial estimates for the unique variances must be given.

GetFactorLoadings
public double[,] GetFactorLoadings()

Description
Returns the unrotated factor loadings.

Returns
A double matrix containing the unrotated factor loadings.

Exceptions
- Imsl.Stat.RankException is thrown if the rank of the covariance matrix is less than the number of factors.
- Imsl.Stat.NotSemiDefiniteException is thrown if the Hessian matrix not semi-definite.
- Imsl.Stat.NotPositiveSemiDefiniteException is thrown if the covariance matrix is not positive semi-definite.
- Imsl.Stat.SingularException is thrown if the covariance matrix is singular.
- Imsl.Stat.BadVarianceException is thrown if the input variance is not in the allowed range.
- Imsl.Stat.EigenvalueException is thrown if an error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.
Imsl.Stat.NonPositiveEigenvalueException is thrown if in alpha factor analysis an eigenvalue is not positive. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced, or new initial estimates for the unique variances must be given.

GetParameterUpdates

public double[] GetParameterUpdates()

Description

Returns the parameter updates.

Returns

A double array containing the parameter updates when convergence was reached (or the iterations terminated).

Remarks

The parameter updates are only meaningful for the common factor model. The parameter updates are set to 0.0 for the principal component model.

Exceptions

Imsl.Stat.RankException is thrown if the rank of the covariance matrix is less than the number of factors.
Imsl.Stat.NotSemiDefiniteException is thrown if the Hessian matrix not semi-definite.
Imsl.Stat.NotPositiveSemiDefiniteException is thrown if the covariance matrix is not positive semi-definite.
Imsl.Stat.SingularException is thrown if the covariance matrix is singular.
Imsl.Stat.BadVarianceException is thrown if the input variance is not in the allowed range.
Imsl.Stat.EigenvalueException is thrown if an error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.
Imsl.Stat.NonPositiveEigenvalueException is thrown if in alpha factor analysis an eigenvalue is not positive. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced, or new initial estimates for the unique variances must be given.

GetPercents

public double[] GetPercents()

Description

Returns the cumulative percent of the total variance explained by each principal component.

Returns

A double array containing the total variance explained by each principal component.

Remarks

Valid for the principal component model.
Exceptions

- `Imsl.Stat.RankException` is thrown if the rank of the covariance matrix is less than the number of factors.
- `Imsl.Stat.NotSemiDefiniteException` is thrown if the Hessian matrix is not semi-definite.
- `Imsl.Stat.NotPositiveSemiDefiniteException` is thrown if the covariance matrix is not positive semi-definite.
- `Imsl.Stat.SingularException` is thrown if the covariance matrix is singular.
- `Imsl.Stat.BadVarianceException` is thrown if the input variance is not in the allowed range.
- `Imsl.Stat.EigenvalueException` is thrown if an error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.
- `Imsl.Stat.NonPositiveEigenvalueException` is thrown if in alpha factor analysis an eigenvalue is not positive. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced, or new initial estimates for the unique variances must be given.

GetStandardErrors

```java
public double[] GetStandardErrors()
```

Description

Returns the estimated asymptotic standard errors of the eigenvalues.

Returns

A `double` array containing the estimated asymptotic standard errors of the eigenvalues.

Exceptions

- `Imsl.Stat.RankException` is thrown if the rank of the covariance matrix is less than the number of factors.
- `Imsl.Stat.NotSemiDefiniteException` is thrown if the Hessian matrix is not semi-definite.
- `Imsl.Stat.NotPositiveSemiDefiniteException` is thrown if the covariance matrix is not positive semi-definite.
- `Imsl.Stat.SingularException` is thrown if the covariance matrix is singular.
- `Imsl.Stat.BadVarianceException` is thrown if the input variance is not in the allowed range.
- `Imsl.Stat.EigenvalueException` is thrown if an error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.
- `Imsl.Stat.NonPositiveEigenvalueException` is thrown if in alpha factor analysis an eigenvalue is not positive. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced, or new initial estimates for the unique variances must be given.

GetStatistics

```java
public double[] GetStatistics()
```

Description

Returns statistics.

Multivariate Analysis  FactorAnalysis ● 1029
Returns
A double array containing output statistics.

Remarks
Statistics are not defined and set to NaN when the method used to obtain the estimates is the principal component method, principal factor method, image factor analysis method, or alpha analysis method.

<table>
<thead>
<tr>
<th></th>
<th>Statistics[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Value of the function minimum.</td>
</tr>
<tr>
<td>1</td>
<td>Tucker reliability coefficient.</td>
</tr>
<tr>
<td>2</td>
<td>Chi-squared test statistic for testing that the number of factors in the model are adequate for the data.</td>
</tr>
<tr>
<td>3</td>
<td>Degrees of freedom in chi-squared. This is computed as $((nvar - nfactors)^2 - nvar - nfactors)/2$ where $nvar$ is the number of variables and $nfactors$ is the number of factors in the model.</td>
</tr>
<tr>
<td>4</td>
<td>Probability of a greater chi-squared statistic.</td>
</tr>
<tr>
<td>5</td>
<td>Number of iterations.</td>
</tr>
</tbody>
</table>

Exceptions
Imsl.Stat.RankException is thrown if the rank of the covariance matrix is less than the number of factors.
Imsl.Stat.NotSemiDefiniteException is thrown if the Hessian matrix not semi-definite.
Imsl.Stat.NotPositiveSemiDefiniteException is thrown if the covariance matrix is not positive semi-definite.
Imsl.Stat.SingularException is thrown if the covariance matrix is singular.
Imsl.Stat.BadVarianceException is thrown if the input variance is not in the allowed range.
Imsl.Stat.EigenvalueException is thrown if an error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.
Imsl.Stat.NonPositiveEigenvalueException is thrown if in alpha factor analysis an eigenvalue is not positive. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced, or new initial estimates for the unique variances must be given.

GetValues
public double[] GetValues()

Description
Returns the eigenvalues.

Returns
A double array containing the eigenvalues of the matrix from which the factors were extracted ordered from largest to smallest.
Remarks
If Alpha Factor analysis is used, then the first nfactors positions of the array contain the Alpha coefficients. Here, nfactors is the number of factors in the model. If the algorithm fails to converge for a particular eigenvalue, that eigenvalue is set to NaN. Note that the eigenvalues are usually not the eigenvalues of the input matrix cov. They are the eigenvalues of the input matrix cov when the Principal Component method is used.

Exceptions
- `Imsl.Stat.RankException` is thrown if the rank of the covariance matrix is less than the number of factors.
- `Imsl.Stat.NotSemiDefiniteException` is thrown if the Hessian matrix is not semi-definite.
- `Imsl.Stat.NotPositiveSemiDefiniteException` is thrown if the covariance matrix is not positive semi-definite.
- `Imsl.Stat.SingularException` is thrown if the covariance matrix is singular.
- `Imsl.Stat.BadVarianceException` is thrown if the input variance is not in the allowed range.
- `Imsl.Stat.EigenvalueException` is thrown if an error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.
- `Imsl.Stat.NonPositiveEigenvalueException` is thrown if in alpha factor analysis an eigenvalue is not positive. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced, or new initial estimates for the unique variances must be given.

GetVariances

```java
public double[] GetVariances()
```

Description
Returns the unique variances.

Returns
A `double` array containing the unique variances.

Exceptions
- `Imsl.Stat.RankException` is thrown if the rank of the covariance matrix is less than the number of factors.
- `Imsl.Stat.NotSemiDefiniteException` is thrown if the Hessian matrix is not semi-definite.
- `Imsl.Stat.NotPositiveSemiDefiniteException` is thrown if the covariance matrix is not positive semi-definite.
- `Imsl.Stat.SingularException` is thrown if the covariance matrix is singular.
- `Imsl.Stat.BadVarianceException` is thrown if the input variance is not in the allowed range.
- `Imsl.Stat.EigenvalueException` is thrown if an error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.
- `Imsl.Stat.NonPositiveEigenvalueException` is thrown if in alpha factor analysis an eigenvalue is not positive. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced, or new initial estimates for the unique variances must be given.
GetVectors

public double[,] GetVectors()

Description

Returns the eigenvectors.

Returns

A double matrix containing the eigenvectors of the matrix from which the factors were extracted.

Remarks

The j-th column of the eigenvector matrix corresponds to the j-th eigenvalue. The eigenvectors are normalized to each have Euclidean length equal to one. Also, the sign of each vector is set so that the largest component in magnitude (the first of the largest if there are ties) is made positive. Note that the eigenvectors are usually not the eigenvectors of the input matrix cov. They are the eigenvectors of the input matrix cov when the Principal Component method is used.

Exceptions

Imsl.Stat.RankException is thrown if the rank of the covariance matrix is less than the number of factors.
Imsl.Stat.NotSemiDefiniteException is thrown if the Hessian matrix not semi-definite.
Imsl.Stat.NotPositiveSemiDefiniteException is thrown if the covariance matrix is not positive semi-definite.
Imsl.Stat.SingularException is thrown if the covariance matrix is singular.
Imsl.Stat.BadVarianceException is thrown if the input variance is not in the allowed range.
Imsl.Stat.EigenvalueException is thrown if an error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.
Imsl.Stat.NonPositiveEigenvalueException is thrown if in alpha factor analysis an eigenvalue is not positive. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced, or new initial estimates for the unique variances must be given.

SetVariances

public void SetVariances(double[] uniq)

Description

Sets the unique variances.

Parameter

uniq – A double array of length nvar containing the unique variances.

Remarks

If this member function is not called, the elements of uniq are set to 0.0. If the iterative methods fail for the unique variances used, new initial estimates should be tried. These may be obtained by use of another factoring method (use the final estimates from the new method as initial estimates in the old method). Another alternative is to call member function VarianceEstimationMethod and set the input argument to 0. This will cause the initial unique variances to be estimated by the code.
Example: Principal Components

This example illustrates the use of the FactorAnalysis class for a nine-variable matrix. FactorAnalysis.Model.PrincipalComponent and input matrix type FactorAnalysis.MatrixType.Correlation are selected.

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;

public class FactorAnalysisEx1
{
    public static void Main(String[] args)
    {
        double[,] corr = {
            {1.0, 0.523, 0.395, 0.471, 0.346, 0.426, 0.576, 0.434, 0.639},
            {0.523, 1.0, 0.479, 0.506, 0.418, 0.462, 0.547, 0.283, 0.645},
            {0.395, 0.479, 1.0, 0.355, 0.27, 0.254, 0.452, 0.219, 0.504},
            {0.471, 0.506, 0.355, 1.0, 0.691, 0.791, 0.443, 0.285, 0.505},
            {0.346, 0.418, 0.27, 0.691, 1.0, 0.679, 0.383, 0.149, 0.409},
            {0.426, 0.462, 0.254, 0.791, 0.679, 1.0, 0.372, 0.314, 0.472},
            {0.576, 0.547, 0.452, 0.443, 0.383, 0.372, 1.0, 0.385, 0.68},
            {0.434, 0.283, 0.219, 0.285, 0.149, 0.314, 0.385, 1.0, 0.47},
            {0.639, 0.645, 0.504, 0.505, 0.409, 0.472, 0.68, 0.47, 1.0}};

        FactorAnalysis pc = new FactorAnalysis(corr,
            FactorAnalysis.MatrixType.Correlation, 9);
        pc.FactorLoadingEstimationMethod = FactorAnalysis.Model.PrincipalComponent;
        pc.DegreesOfFreedom = 100;

        PrintMatrixFormat pmf = new PrintMatrixFormat();
        pmf.NumberFormat = "0.0000";
        new PrintMatrix("Eigenvalues").Print(pmf, pc.GetValues());
        new PrintMatrix("Percents").Print(pmf, pc.GetPercents());
        new PrintMatrix("Standard Errors").Print(pmf, pc.GetStandardErrors());
        new PrintMatrix("Eigenvectors").Print(pmf, pc.GetVectors());
        new PrintMatrix("Unrotated Factor Loadings").Print(pmf, pc.GetFactorLoadings());
    }
}
```

### Output

#### Eigenvalues

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.6769</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.2640</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.8444</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.5550</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.4471</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.4291</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.3102</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.2770</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.1962</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Percents

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5197</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.6601</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.7539</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.8156</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.8653</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.9130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.9474</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.9782</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Standard Errors

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6498</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.1771</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.0986</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.0879</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.0882</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.0890</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.0944</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.0994</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.1113</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Eigenvectors

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.3462</td>
<td>-0.2354</td>
<td>0.1386</td>
<td>-0.3317</td>
<td>-0.1088</td>
<td>0.7974</td>
<td>0.1735</td>
<td>-0.1240</td>
<td>-0.0488</td>
</tr>
<tr>
<td>1</td>
<td>0.3526</td>
<td>-0.1108</td>
<td>-0.2795</td>
<td>-0.2161</td>
<td>0.7664</td>
<td>-0.2002</td>
<td>0.1386</td>
<td>-0.3032</td>
<td>-0.0079</td>
</tr>
<tr>
<td>2</td>
<td>0.2754</td>
<td>-0.2697</td>
<td>-0.5585</td>
<td>0.6939</td>
<td>-0.1531</td>
<td>0.1511</td>
<td>0.0099</td>
<td>-0.0406</td>
<td>-0.0997</td>
</tr>
<tr>
<td>3</td>
<td>0.3664</td>
<td>0.4031</td>
<td>0.0406</td>
<td>0.1196</td>
<td>0.0017</td>
<td>0.1152</td>
<td>-0.4022</td>
<td>-0.1178</td>
<td>0.7060</td>
</tr>
<tr>
<td>4</td>
<td>0.3144</td>
<td>0.5022</td>
<td>-0.0733</td>
<td>-0.0207</td>
<td>-0.2804</td>
<td>-0.1796</td>
<td>0.7295</td>
<td>0.0075</td>
<td>0.0046</td>
</tr>
<tr>
<td>5</td>
<td>0.3455</td>
<td>0.4553</td>
<td>0.1825</td>
<td>0.1114</td>
<td>0.1202</td>
<td>0.0696</td>
<td>-0.3742</td>
<td>0.0925</td>
<td>-0.6780</td>
</tr>
<tr>
<td>6</td>
<td>0.3487</td>
<td>-0.2714</td>
<td>-0.0725</td>
<td>-0.3545</td>
<td>-0.5242</td>
<td>-0.4355</td>
<td>-0.2854</td>
<td>-0.3408</td>
<td>-0.1089</td>
</tr>
<tr>
<td>7</td>
<td>0.2407</td>
<td>-0.3159</td>
<td>0.7383</td>
<td>0.4329</td>
<td>0.0861</td>
<td>-0.1969</td>
<td>0.1862</td>
<td>-0.1623</td>
<td>0.0505</td>
</tr>
<tr>
<td>8</td>
<td>0.3847</td>
<td>-0.2533</td>
<td>-0.0078</td>
<td>-0.1468</td>
<td>0.0459</td>
<td>-0.1498</td>
<td>-0.0251</td>
<td>0.8521</td>
<td>0.1225</td>
</tr>
</tbody>
</table>

#### Unrotated Factor Loadings

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.7487</td>
<td>-0.2646</td>
<td>0.1274</td>
<td>-0.2471</td>
<td>-0.0728</td>
<td>0.5224</td>
<td>0.0966</td>
<td>-0.0652</td>
<td>-0.0216</td>
</tr>
<tr>
<td>1</td>
<td>0.7625</td>
<td>-0.1245</td>
<td>-0.2568</td>
<td>-0.1610</td>
<td>0.5124</td>
<td>-0.1312</td>
<td>0.0772</td>
<td>-0.1596</td>
<td>-0.0035</td>
</tr>
<tr>
<td>2</td>
<td>0.5956</td>
<td>-0.3032</td>
<td>-0.5133</td>
<td>0.5170</td>
<td>-0.1024</td>
<td>0.0990</td>
<td>0.0055</td>
<td>-0.0214</td>
<td>-0.0442</td>
</tr>
<tr>
<td>3</td>
<td>0.7923</td>
<td>0.4532</td>
<td>0.0373</td>
<td>0.0891</td>
<td>0.0012</td>
<td>0.0755</td>
<td>-0.2240</td>
<td>-0.0620</td>
<td>0.3127</td>
</tr>
</tbody>
</table>
Example: Factor Analysis

This example illustrates the use of the FactorAnalysis class. The following data were originally analyzed by Emmett (1949). There are 211 observations on 9 variables. Following Lawley and Maxwell (1971), three factors will be obtained by the method of maximum likelihood.

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;

public class FactorAnalysisEx2
{
    public static void Main(String[] args)
    {
        double[,] cov = {
            {1.0, 0.523, 0.395, 0.471, 0.346, 0.426, 0.576, 0.434, 0.639},
            {0.523, 1.0, 0.479, 0.506, 0.418, 0.462, 0.547, 0.283, 0.645},
            {0.395, 0.479, 1.0, 0.355, 0.27, 0.254, 0.452, 0.219, 0.504},
            {0.471, 0.506, 0.355, 1.0, 0.691, 0.791, 0.443, 0.285, 0.505},
            {0.346, 0.418, 0.27, 0.691, 1.0, 0.679, 0.383, 0.149, 0.409},
            {0.426, 0.462, 0.254, 0.791, 0.679, 1.0, 0.372, 0.314, 0.472},
            {0.576, 0.547, 0.452, 0.443, 0.383, 0.372, 1.0, 0.385, 0.68},
            {0.434, 0.283, 0.219, 0.285, 0.149, 0.314, 0.385, 1.0, 0.47},
            {0.639, 0.645, 0.504, 0.505, 0.409, 0.472, 0.68, 0.47, 1.0}};

        FactorAnalysis fl = new FactorAnalysis(cov,
            FactorAnalysis.MatrixType.VarianceCovariance, 3);
        fl.ConvergenceCriterion1 = .000001;
        fl.ConvergenceCriterion2 = .01;
        fl.VarianceEstimationMethod = 0;
        fl.MaxStep = 10;
        fl.DegreesOfFreedom = 210;

        PrintMatrixFormat pmf = new PrintMatrixFormat();
        pmf.NumberFormat = "0.0000";
        new PrintMatrix
            ("Unique Error Variances").Print(pmf, fl.GetVariances());
    }
}
```
new PrintMatrix
    ("Unrotated Factor Loadings").Print(pmf, fl.GetFactorLoadings());
new PrintMatrix("Eigenvalues").Print(pmf, fl.GetValues());
new PrintMatrix("Statistics").Print(pmf, fl.GetStatistics());

} // End of output

Output

Unique Error Variances
  0  0.4505
  1  0.4271
  2  0.6166
  3  0.2123
  4  0.3805
  5  0.1769
  6  0.3995
  7  0.4615
  8  0.2509

Unrotated Factor Loadings
  0  1  2
  0  0.6642 -0.3209  0.0735
  1  0.6888 -0.2471 -0.1933
  2  0.4926 -0.3022 -0.2224
  3  0.8372  0.2924 -0.0354
  4  0.7050  0.3148 -0.0354
  5  0.8187  0.3767  0.1045
  6  0.6615 -0.3960 -0.0777
  7  0.4579 -0.2955  0.4913
  8  0.7657 -0.4274 -0.0117

Eigenvalues
  0  0.0626
  1  0.2295
  2  0.5413
  3  0.8650
  4  0.8937
  5  0.9736
  6  1.0802
  7  1.1172
  8  1.1401

Statistics
  0  0.0350
  1  1.0000
  2  7.1494
  3  12.0000
  4  0.8476
  5  5.0000

1036 • FactorAnalysis

IMSL C# Numerical Library
FactorAnalysis.MatrixType Enumeration

public enumeration Imsl.Stat.FactorAnalysis.MatrixType

Matrix type.

Fields

Correlation
public Imsl.Stat.FactorAnalysis.MatrixType Correlation

Description
Indicates correlation matrix.

VarianceCovariance
public Imsl.Stat.FactorAnalysis.MatrixType VarianceCovariance

Description
Indicates variance-covariance matrix.

FactorAnalysis.Model Enumeration

public enumeration Imsl.Stat.FactorAnalysis.Model

Model type.

Fields

AlphaFactorAnalysis

Description
Indicates alpha-factor analysis (common factor model) method used to obtain the estimates. Degrees of freedom is used for this estimation method.

GeneralizedLeastSquares


Description
Indicates generalized least-squares (common factor model) method used to obtain the estimates.

ImageFactorAnalysis

Description
Indicates Image-factor analysis (common factor model) method used to obtain the estimates.

MaximumLikelihood

Description
Indicates maximum likelihood method used to obtain the estimates. Degrees of freedom is used for this estimation method.

PrincipalComponent

Description
Indicates principal component (principal component model) used to obtain the estimates.

PrincipalFactor
public Imsl.Stat.FactorAnalysis.Model PrincipalFactor

Description
Indicates principal factor (common factor model) will be used to obtain the estimates.

UnweightedLeastSquares

Description
Indicates unweighted least-squares (common factor model) method used to obtain the estimates. This option is the default.

DiscriminantAnalysis Class

public class Imsl.Stat.DiscriminantAnalysis
Performs a linear or a quadratic discriminant function analysis among several known groups.

DiscriminantAnalysis allows linear or a quadratic discrimination and the use of either reclassification, split sample, or the leaving-out-one methods in order to evaluate the rule. One or more observations can be added to the rule during each invocation of the Update method.
DiscriminantAnalysis results in the measure of distance between the groups, (see GetMahalanobis method), a table summarizing the classification results, (see GetClassTable), a matrix containing the posterior probabilities of group membership for each classified observation, (see GetProbability), the within-sample means, (see GetMeans) and covariance matrices computed from their LU factorizations, (see GetCovariance). The linear discriminant function coefficients are also computed, (see GetCoefficients method).

All observations can be input during one call to the Update method; this has the advantage of simplicity. Alternatively, one or more rows of observations can be input during separate calls to Update. This does not require all observations be memory resident, a significant advantage with large data sets. Note, however, to classify the same data set requires a second pass of the data to the Classify method. During the first pass to the Update method the discriminant functions are computed while in the second pass to the Classify method the observations are classified. When known groups are available the method GetClassTable is useful in comparing how well the algorithm classifies. Multiple calls to the Classify method are also allowed. The class table, GetClassTable, is an accumulation of all observations classified. The class membership and probabilities, returned in GetClassMembership and GetProbability, will contain the membership for each observation from the most recent invocation of the Classify method.

Pooled only and pooled with group covariance computation cannot be mixed. By default, both pooled and group covariance matrices will be computed. An InvalidOperationException will be thrown if an attempt is made to change the covariance computation after the first call to the Update method. See the CovarianceComputation method for more details on specifying the covariance computation.

The within-group means are updated for all valid observations in x. Observations with invalid group numbers are ignored, as are observations with missing values (Double.NaN). The LU factorization of the covariance matrices are updated by adding (or deleting) observations via Givens rotations. See the Downdate method to delete observations.

During the algorithm’s training process, or each invocation of the Update method, each observation in x is added to the means and the factorizations of the covariance matrices. Statistics of interest are computed: the linear discriminant functions, the prior probabilities, the log of the determinant of each of the covariance matrices, and a test statistic for testing that all of the within-group covariance matrices are equal. The matrix of Mahalanobis distances, which consists of the distances between the groups, is computed via the pooled covariance matrix when linear discrimination is specified. The row covariance matrix is used when the discrimination is quadratic. Covariance matrices are defined as follows. Let \( N_i \) denote the sum of the frequencies of the observations in group \( i \), and let \( M_i \) denote the number of observations in group \( i \). Then, if \( S_i \) denotes the within-group \( i \) covariance matrix,

\[
S_i = \frac{1}{N_i - 1} \sum_{j=1}^{M_i} w_j f_j (x_j - \bar{x})(x_j - \bar{x})^T
\]

where \( w_j \) is the weight of the \( j \)-th observation in group \( i \), \( f_j \) is its frequency, \( x_j \) is the \( j \)-th observation column vector (in group \( i \)), and \( \bar{x} \) denotes the mean vector of the observations in group \( i \). The mean vectors are computed as

\[
\bar{x} = \frac{1}{W_i} \sum_{j=1}^{M_i} w_j f_j x_j
\]
where

\[ W_i = \sum_{j=1}^{M_i} w_j f_j \]

Given the means and the covariance matrices, the linear discriminant function for group \( i \) is computed as:

\[ z_i = \ln(p_i) - 0.5 x^T S_p^{-1} x + x^T S_p^{-1} \bar{x}_i \]

where \( \ln(p_i) \) is the natural log of the prior probability for the \( i \)-th group, \( x \) is the observation to be classified, and \( S_p \) denotes the pooled covariance matrix.

Let \( S \) denote either the pooled covariance matrix or one of the within-group covariance matrices \( S_i \). (\( S \) will be the pooled covariance matrix in linear discrimination, and \( S_i \) otherwise.) The Mahalanobis distance between group \( i \) and group \( j \) is computed as:

\[ D^2_{ij} = (x_i - x_j)^T S^{-1} (x_i - x_j) \]

Finally, the asymptotic chi-squared test for the equality of covariance matrices is computed as follows (Morrison 1976, page 252):

\[ \gamma = C^{-1} \sum_{i=1}^{k} n_i \{ \ln(|S_p|) - \ln(|S_i|) \} \]

where \( n_i \) is the number of degrees of freedom in the \( i \)-th sample covariance matrix, \( k \) is the number of groups, and

\[ C^{-1} = \frac{1 - 2p^2 + 3p - 1}{6(p+1)(k-1)} \left( \sum_{i=1}^{k} \frac{1}{n_i} - \frac{1}{\sum_{j=1}^{k} n_j} \right) \]

where \( p \) is the number of variables.

The estimated posterior probability of each observation \( x \) belonging to group \( i \) is computed using the prior probabilities and the sample mean vectors and estimated covariance matrices under a multivariate normal assumption. Under quadratic discrimination, the within-group covariance matrices are used to compute the estimated posterior probabilities. The estimated posterior probability of an observation \( x \) belonging to group \( i \) is

\[ \hat{q}_i(x) = \frac{e^{-\frac{1}{2} D^2_i(x)}}{\sum_{k=1}^{k} e^{-\frac{1}{2} D^2_j(x)}} \]

where

\[ D^2_i(x) = \begin{cases} (x - \bar{x}_i)^T S_i^{-1} (x - \bar{x}_i) + \ln |S_i| - 2 \ln(p_i) & \text{linear or quadratic, pooled, group} \\ (x - \bar{x}_i)^T S_p^{-1} (x - \bar{x}_i) - 2 \ln(p_i) & \text{linear, pooled} \end{cases} \]

For the leaving-out-one method of classification, the sample mean vector and sample covariance matrices in the formula for

\[ D^2_i(x) \]

are adjusted so as to remove the observation \( x \) from their computation. For linear discrimination, the linear discriminant function coefficients are actually used to compute the same posterior probabilities.
Using the posterior probabilities, each observation in \( x \) is classified into a group; the result is tabulated in the matrix returned by `GetClassTable` and saved in the vector returned by `GetClassMembership`. If a group variable is provided and the group number is out of range, the classification table is not altered at this stage. If the reclassification method is specified, then all observations with no missing values are classified. When the leaving-out-one method is used, observations with invalid group numbers, weights, frequencies or classification variables are not classified. Regardless of the frequency, a 1 is added (or subtracted) from the classification table for each row of \( x \) that is classified and contains a valid group number. When the leaving-out-one method is used, adjustment is made to the posterior probabilities to remove the effect of the observation in the classification rule. In this adjustment, each observation is presumed to have a weight of \( w_j \) and a frequency of 1.0. See Lachenbruch (1975, page 36) for the required adjustment.

**Properties**

**ClassificationMethod**

```csharp
public Imsl.Stat.DiscriminantAnalysis.Classification ClassificationMethod { get; set; }
```

**Description**
The classification method.

**Property Value**
Indicates the method of classification.

Default: `Classification.Reclassification` is used.

**Remarks**
Use `Classification.Reclassification` or `Classification.LeaveOutOne`.

**CovarianceComputation**

```csharp
public Imsl.Stat.DiscriminantAnalysis.CovarianceMatrix CovarianceComputation { get; set; }
```

**Description**
The type of covariance matrices to be computed.

**Property Value**
Indicates the type of covariance matrices to be computed.

Default: `CovarianceMatrix.PooledGroup` is used.

**Remarks**

**DiscriminationMethod**

```csharp
public Imsl.Stat.DiscriminantAnalysis.Discrimination DiscriminationMethod { get; set; }
```
**Description**
The discrimination method.

**Property Value**
Indicates the method of discrimination.
Default: Discrimination.Linear is used.

**Remarks**
Use Discrimination.Linear or Discrimination.Quadratic.

**NumberOfRowsMissing**
`public int NumberOfRowsMissing {get; }

Description
The number of rows of data encountered containing missing values (Double.NaN).

**Property Value**
An int representing the number of rows of data encountered containing missing values (Double.NaN) for the classification, group, weight, and/or frequency variables.

**Remarks**
If a row of data contains a missing value (Double.NaN) for any of these variables, that row is excluded from the computations.

**PriorType**
`public Imsl.Stat.DiscriminantAnalysis.PriorProbabilities PriorType {get; set; }

Description
The type of prior probabilities to be calculated.

**Property Value**
Indicates the type of prior probabilities to be calculated.
Default: PriorType = PriorProbabilities.Equal.

**Remarks**
Use PriorProbabilities.Equal to set equal prior probabilities, calculated as 1.0/nGroups. Or, use PriorProbabilities.Proportional to calculate the priors to be proportional to the sample size in each group. The sum of all prior probabilities is equal to 1.0. If the values calculated for the priors are less than 1.0e-20, they will be converted to the Math.Log(1.0e-20). Prior probabilities are used in calculating statistics, coefficients, Mahalanobis, and classification probabilities.

**Constructor**

**DiscriminantAnalysis**
`public DiscriminantAnalysis(int nVariables, int nGroups)
Description
Constructs a DiscriminantAnalysis.

Parameters
nVariables – An int representing the number of variables to be used in the discrimination.
nGroups – An int representing the number of groups in the data.

Methods

Classify
public void Classify(double[,] x)

Description
Classify a set of observations using the linear or quadratic discriminant functions generated during the training process.

Parameter
x – A double matrix containing the observations with at least nVariables columns. The first nVariables columns correspond to the variables. Reclassification does not require group numbers be present. Any additional columns will be ignored.

Remarks
An InvalidOperationException is thrown if the leave-out-one classification method is chosen.

Exceptions
Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.
Imsl.Stat.EmptyGroupException is thrown when there are no observations in a group.
Imsl.Stat.CovarianceSingularException is thrown when the variance-covariance matrix is singular.

Classify
public void Classify(double[,] x, int[] varIndex)

Description
Classify a set of observations using the linear or quadratic discriminant functions generated during the training process.

Parameters
x – A double matrix containing the observations with at least nVariables columns. The columns indicated in varIndex correspond to the variables. Reclassification does not require group numbers be present. Additional columns will be ignored.
varIndex – An int array containing the column indices in x that correspond to the variables to be used in the analysis.
Remarks
An InvalidOperationException is thrown if the leave-out-one classification method is chosen.

Exceptions

- Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.
- Imsl.Stat.EmptyGroupException is thrown when there are no observations in a group.
- Imsl.Stat.CovarianceSingularException is thrown when the variance-covariance matrix is singular.

Classify

```csharp
public void Classify(double[,] x, int[] frequencies, double[] weights)
```

Description

Classify a set of observations and associated frequencies and weights using the linear or quadratic discriminant functions generated during the training process.

Parameters

- `x` – A double matrix containing the observations with at least `nVariables` columns. The first `nVariables` columns correspond to the variables. Reclassification does not require group numbers to be present. Any additional columns will be ignored.
- `frequencies` – An int array containing the associated frequencies for each observation.
- `weights` – A double array containing the associated weights for each observation

Remarks

An InvalidOperationException is thrown if the leave-out-one classification method is chosen.

Exceptions

- Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.
- Imsl.Stat.EmptyGroupException is thrown when there are no observations in a group.
- Imsl.Stat.CovarianceSingularException is thrown when the variance-covariance matrix is singular.

Classify

```csharp
public void Classify(double[,] x, int[] frequencies, double[] weights)
```

Description

Classify a set of observations and associated frequencies and weights using the linear or quadratic discriminant functions generated during the training process.
Parameters

- **x** – A double matrix containing the observations with at least nVariables columns. The columns indicated in varIndex correspond to the variables. Reclassification does not require group numbers be present. Additional columns in x will be ignored.
- **varIndex** – An int array containing the column indices in x that correspond to the variables to be used in the analysis.
- **frequencies** – An int array containing the associated frequencies for each observation.
- **weights** – A double array containing the associated weights for each observation.

Remarks

An InvalidOperationException is thrown if the leave-out-one classification method is chosen.

Exceptions

- Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.
- Imsl.Stat.EmptyGroupException is thrown when there are no observations in a group.
- Imsl.Stat.CovarianceSingularException is thrown when the variance-covariance matrix is singular.

Classify

```csharp
class DiscriminantAnalysis
{
    public void Classify(double[,] x, int[] group, int[] varIndex)
{
    
Description

Classify a set of observations and compare against known groups using the linear or quadratic discriminant functions generated during the training process.

Parameters

- **x** – A double matrix containing the observations with at least nVariables columns. The columns indicated in varIndex correspond to the variables. Any additional columns will be ignored.
- **group** – An int array containing the group numbers. The groups must be numbered 1,2, ..., nGroups for each observation.
- **varIndex** – An int array containing the column indices in x that correspond to the variables to be used in the analysis.

Exceptions

- Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.
- Imsl.Stat.EmptyGroupException is thrown when there are no observations in a group.
- Imsl.Stat.CovarianceSingularException is thrown when the variance-covariance matrix is singular.

Classify

```csharp
public void Classify(double[,] x, int[] group, int[] varIndex, int[] frequencies, double[] weights)
```
**Description**
Classify a set of observations, associated frequencies and weights, and compare against known groups using the linear or quadratic discriminant functions generated during the training process.

**Parameters**
- **x** – A double matrix containing the observations with at least nVariables columns. The columns indicated in varIndex correspond to the variables. Additional columns are ignored.
- **group** – An int array containing the group numbers. The groups must be numbered 1, 2, ..., nGroups for each observation.
- **varIndex** – An int array containing the column indices in x that correspond to the variables to be used in the analysis.
- **frequencies** – An int array containing the associated frequencies for each observation.
- **weights** – A double array containing the associated weights for each observation.

**Exception**
Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.

**Downdate**
public void Downdate(double[,] x, int[] group)

**Description**
Removes a set of observations from the discriminant functions.

**Parameters**
- **x** – A double matrix containing the observations to be removed, with at least nVariables columns. The first nVariables columns correspond to the variables. Any additional columns will be ignored.
- **group** – An int array containing the group numbers. The groups must be numbered 1, 2, ..., nGroups for each observation.

**Exception**
Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.
Parameters

\( x \) – A double matrix containing the observations to be removed, with at least \( n\text{Variables} \) columns. The columns indicated in \( \text{varIndex} \) correspond to the variables.

\( \text{group} \) – An int array containing the group numbers. The groups must be numbered 1,2, ..., \( n\text{Groups} \) for each observation.

\( \text{varIndex} \) – An int array containing the column indices in \( x \) that correspond to the variables to be used in the analysis.

Exception

\text{Imsl.Stat.SumOfWeightsNegException} \) is thrown when the sum of the weights have become negative.

\textbf{Downdate}

\texttt{public void Downdate(double[,] x, int[] group, int[] frequencies, double[] weights)}

\textbf{Description}

Removes a set of observations and associated frequencies and weights from the discriminant functions.

Parameters

\( x \) – A double matrix containing the observations to be removed, with at least \( n\text{Variables} \) columns. The columns indicated in \( \text{varIndex} \) correspond to the variables.

\( \text{group} \) – An int array containing the group numbers. The groups must be numbered 1,2, ..., \( n\text{Groups} \) for each observation.

\( \text{frequencies} \) – An int array containing the associated frequencies for each observation.

\( \text{weights} \) – A double array containing the associated weights for each observation.

Exception

\text{Imsl.Stat.SumOfWeightsNegException} \) is thrown when the sum of the weights have become negative.

\textbf{Downdate}

\texttt{public void Downdate(double[,] x, int[] group, int[] \text{varIndex}, int[] frequencies, double[] weights)}

\textbf{Description}

Removes a set of observations and associated frequencies and weights from the discriminant functions.

Parameters

\( x \) – A double matrix containing the observations to be removed, with at least \( n\text{Variables} \) columns. The columns indicated in \( \text{varIndex} \) correspond to the variables.

\( \text{group} \) – An int array containing the group numbers. The groups must be numbered 1,2, ..., \( n\text{Groups} \) for each observation.

\( \text{varIndex} \) – An int array containing the column indices in \( x \) that correspond to the variables to be used in the analysis.
frequencies – An int array containing the associated frequencies for each observation.
weights – A double array containing the associated weights for each observation.

Exception
Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.

GetClassMembership
public int[] GetClassMembership()

Description
Returns the group number to which the observation was classified.

Returns
An int array containing the group to which the observation was classified. If an observation has an invalid group number, frequency, or weight when the leaving-out-one method has been specified, then the observation is not classified and the corresponding elements of the array are set to zero. Note this will return the class membership of the last set of observations classified.

Remarks
An InvalidOperationException is thrown if no data has been classified.

GetClassTable
public double[,] GetClassTable()

Description
Returns the classification table.

Returns
An nGroups by nGroups double matrix containing the classification table. The accumulation of each observation that is classified and has a group number equal to 1, 2, ..., nGroups is entered into the table. If a known group is provided, the rows of the table correspond to the known group membership. The columns refer to the group to which the observation was classified. If a known group is not provided, the table will only contain the accumulated classified groups in the column corresponding to the group to which the observation was classified.

Remarks
An InvalidOperationException is thrown if no data has been classified.

GetCoefficients
public double[,] GetCoefficients()

Description
Returns the linear discriminant function coefficients.

Returns
An nGroups by nVariables double matrix containing the linear discriminant function coefficients. The first column of the matrix contains the constant term, and the remaining columns contain the variable coefficients. The i-th row of the returned matrix corresponds to group i. The coefficients are always computed as linear discriminant function coefficients even when quadratic discrimination is specified.
**Exceptions**

- `Imsl.Stat.EmptyGroupException` is thrown when there are no observations in a group.
- `Imsl.Stat.CovarianceSingularException` is thrown when the variance-covariance matrix is singular.

**GetCovariance**

```java
public double[, ,] GetCovariance()
```

**Description**

Returns the array of covariances.

**Returns**

A $g$ by $n\text{Variables}$ by $n\text{Variables}$ double array containing the covariances. Where, $g = n\text{Groups}+1$ if pooled, group covariance computation is specified or $g=1$ if pooled covariance computation is specified. When pooled only covariance matrices are computed, the within-group covariance matrices are not computed. The pooled covariance matrix is always computed and is returned as the $g$-th covariance matrix.

If this method is invoked before classification, the unscaled covariance matrix will be returned.

**Exceptions**

- `Imsl.Stat.EmptyGroupException` is thrown when there are no observations in a group.
- `Imsl.Stat.CovarianceSingularException` is thrown when the variance-covariance matrix is singular.

**GetGroupCounts**

```java
public int[] GetGroupCounts()
```

**Description**

Returns the group counts.

**Returns**

An int array of length $n\text{Groups}$ containing the number of observations in each group. If an update has not preceeded the invocation of this method, an array of all zeros will be returned.

**GetMahalanobis**

```java
public double[, ] GetMahalanobis()
```

**Description**

Returns the Mahalanobis distances between the group means.

**Returns**

An $n\text{Groups}$ by $n\text{Groups}$ double matrix containing the Mahalanobis distances between the group means. For linear discrimination, the Mahalanobis distance

$$D_{ij}^2(x)$$

between group means $i$ and $j$ is computed using the within covariance matrix for group $i$ in place of the pooled covariance matrix.
Exceptions

- `Imsl.Stat.EmptyGroupException` is thrown when there are no observations in a group.
- `Imsl.Stat.CovarianceSingularException` is thrown when the variance-covariance matrix is singular.

---

**GetMeans**

```csharp
public double[,] GetMeans()
```

**Description**

Returns the variable means.

**Returns**

An `nGroups` by `nVariables` `double` matrix containing the variable means. The `i`-th row contains the variable means for group `i`. If this method is invoked before classification, the unscaled means will be returned.

**Exceptions**

- `Imsl.Stat.EmptyGroupException` is thrown when there are no observations in a group.
- `Imsl.Stat.CovarianceSingularException` is thrown when the variance-covariance matrix is singular.

---

**GetPrior**

```csharp
public double[] GetPrior()
```

**Description**

Returns the prior probabilities.

**Returns**

A `double` array of length `nGroups` containing the prior probabilities for each group.

---

**GetProbability**

```csharp
public double[,] GetProbability()
```

**Description**

Returns the posterior probabilities for each observation.

**Returns**

An `nGroups` by `nGroups` `double` matrix containing the posterior probabilities for each observation. Note this will return the probabilities of the last set of observations classified.

**Remarks**

An `InvalidOperationException` is thrown if no data has been classified.

---

**GetStatistics**

```csharp
public double[] GetStatistics()
```

**Description**

Returns statistics.
Returns

A double array containing output statistics. The following table describes the statistics available:

<table>
<thead>
<tr>
<th>index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Sum of the degrees of freedom for the within-covariance matrices.</td>
</tr>
<tr>
<td>1</td>
<td>Chi-squared statistic.</td>
</tr>
<tr>
<td>2</td>
<td>The degrees of freedom in the chi-squared statistic.</td>
</tr>
<tr>
<td>3</td>
<td>Probability of a greater chi-squared, respectively, of a test of the homogeneity of the within-covariance matrices. (Not computed when the pooled only covariance matrix is computed).</td>
</tr>
<tr>
<td>4 thru (4+nGroups)</td>
<td>Log of the determinant of each group’s covariance matrix (not computed when the pooled only covariance matrix is computed) and of the pooled covariance matrix.</td>
</tr>
<tr>
<td>Last (nGroups + 1) elements</td>
<td>Sum of the weights within each group.</td>
</tr>
<tr>
<td>Last element</td>
<td>Sum of the weights in all groups.</td>
</tr>
</tbody>
</table>

Exceptions

Imsl.Stat.EmptyGroupException is thrown when there are no observations in a group.
Imsl.Stat.CovarianceSingularException is thrown when the variance-covariance matrix is singular.

SetPrior

public void SetPrior(double[] prior)

Description

Specifies user supplied prior probabilities.

Parameter

prior – A double array of length nGroups containing the prior probabilities for each group.

Remarks

The elements of prior should sum to 1.0.
If the values of prior are less than 1.0e-20, they will be converted to Math.Log(1.0e-20).
Default: The prior probabilities are calculated to be equal, see property PriorType.

Update

public void Update(double[,] x, int[] group)

Description

Trains a set of observations and associated frequencies and weights by performing a linear or quadratic discriminant function analysis among several known groups.
Parameters

- **x** – A double matrix containing the observations with at least \( n_{\text{Variables}} \) columns. The first \( n_{\text{Variables}} \) correspond to the variables. Any additional columns will be ignored.
- **group** – An int array containing the group numbers. The groups must be numbered 1, 2, ..., \( n_{\text{Groups}} \) for each observation.

Exception

- Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.

Update

```java
public void Update(double[,] x, int[] group, int[] varIndex)
```

Description

Trains a set of observations and associated frequencies and weights by performing a linear or quadratic discriminant function analysis among several known groups.

Parameters

- **x** – A double matrix containing the observations with at least \( n_{\text{Variables}} \) columns. The columns indicated in `varIndex` correspond to the variables. Any additional columns will be ignored.
- **group** – An int array containing the group numbers. The groups must be numbered 1, 2, ..., \( n_{\text{Groups}} \) for each observation.
- **varIndex** – An int array containing the column indices in `x` that correspond to the variables to be used in the analysis.

Exception

- Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.

Update

```java
public void Update(double[,] x, int[] group, int[] frequencies, double[] weights)
```

Description

Trains a set of observations and associated frequencies and weights by performing a linear or quadratic discriminant function analysis among several known groups.

Parameters

- **x** – A double matrix containing the observations with at least \( n_{\text{Variables}} \) columns. The first \( n_{\text{Variables}} \) correspond to the variables. Any additional columns will be ignored.
- **group** – An int array containing the group numbers. The groups must be numbered 1, 2, ..., \( n_{\text{Groups}} \) for each observation.
- **frequencies** – An int array containing the associated frequencies for each observation.
- **weights** – A double array containing the associated weights for each observation.
Exception

Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.

Update

public void Update(double[,] x, int[] group, int[] varIndex, int[] frequencies, double[] weights)

Description

Trains a set of observations and associated frequencies and weights by performing a linear or quadratic
discriminant function analysis among several known groups.

Parameters

- **x** – A double matrix containing the observations with at least `nVariables` columns. The columns
  indicated in `varIndex` correspond to the variables.
- **group** – An int array containing the group numbers. The groups must be numbered
  1, 2, ..., `nGroups` for each observation.
- **varIndex** – An int array containing the column indices in `x` that correspond to the variables to be
  used in the analysis.
- **frequencies** – An int array containing the associated frequencies for each observation.
- **weights** – A double array containing the associated weights for each observation.

Exception

Imsl.Stat.SumOfWeightsNegException is thrown when the sum of the weights have become negative.

Example: Discriminant Analysis

This example uses linear discrimination with equal prior probabilities on Fisher's (1936) iris data. This
example illustrates the use of the DiscriminantAnalysis class.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;

public class DiscriminantAnalysisEx1
{
    public static void Main(String[] args)
    {
        double[,] x = {
            {1.0, 5.1, 3.5, 1.4, .2}, {1.0, 4.9, 3.0, 1.4, .2},
            {1.0, 5.0, 3.6, 1.3, .2}, {1.0, 4.6, 3.1, 1.5, .2},
            {1.0, 4.6, 3.4, 1.4, .3}, {1.0, 5.0, 3.4, 1.5, .2},
            {1.0, 4.4, 2.9, 1.4, .2}, {1.0, 4.9, 3.1, 1.5, .1},
            {1.0, 5.4, 3.7, 1.5, .2}, {1.0, 4.8, 3.4, 1.6, .2},
            {1.0, 4.8, 3.0, 1.4, .1}, {1.0, 4.3, 3.0, 1.1, .1},
        };
```
<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0, 5.8, 4.0, 1.2, .2</td>
<td>1.0, 5.7, 4.4, 1.5, .4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.4, 3.9, 1.3, .4</td>
<td>1.0, 5.1, 3.5, 1.4, .3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.7, 3.8, 1.7, .3</td>
<td>1.0, 5.1, 3.8, 1.5, .3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.4, 3.4, 1.7, .2</td>
<td>1.0, 5.1, 3.7, 1.5, .4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 4.6, 3.6, 1.0, .2</td>
<td>1.0, 5.1, 3.3, 1.7, .5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 4.8, 3.4, 1.9, .2</td>
<td>1.0, 5.0, 3.0, 1.6, .2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.0, 3.4, 1.6, .4</td>
<td>1.0, 5.2, 3.5, 1.5, .2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.2, 3.4, 1.4, .2</td>
<td>1.0, 4.7, 3.2, 1.6, .2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 4.8, 3.1, 1.6, .2</td>
<td>1.0, 5.4, 3.4, 1.5, .4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.2, 4.1, 1.5, .1</td>
<td>1.0, 5.5, 4.2, 1.4, .2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 4.9, 3.1, 1.5, .2</td>
<td>1.0, 5.0, 3.2, 1.2, .2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.5, 3.5, 1.3, .2</td>
<td>1.0, 4.9, 3.6, 1.4, .1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 4.4, 3.0, 1.3, .2</td>
<td>1.0, 5.1, 3.4, 1.5, .2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.0, 3.5, 1.3, .3</td>
<td>1.0, 4.5, 2.3, 1.3, .3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 4.4, 3.2, 1.3, .2</td>
<td>1.0, 5.0, 3.5, 1.6, .6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.1, 3.8, 1.9, .4</td>
<td>1.0, 4.8, 3.0, 1.4, .3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.1, 3.8, 1.6, .2</td>
<td>1.0, 4.6, 3.2, 1.4, .2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 5.3, 3.7, 1.5, .2</td>
<td>1.0, 5.0, 3.3, 1.4, .2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 7.0, 3.2, 4.7, 1.4</td>
<td>2.0, 6.4, 3.2, 4.5, 1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.9, 3.1, 4.9, 1.5</td>
<td>2.0, 5.5, 2.3, 4.0, 1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.5, 2.8, 4.6, 1.5</td>
<td>2.0, 5.7, 2.8, 4.5, 1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.3, 3.3, 4.7, 1.6</td>
<td>2.0, 4.9, 2.4, 3.3, 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.6, 2.9, 4.6, 1.3</td>
<td>2.0, 5.2, 2.7, 3.9, 1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.0, 2.0, 3.5, 1.0</td>
<td>2.0, 5.9, 3.0, 4.2, 1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.0, 2.2, 4.0, 1.0</td>
<td>2.0, 6.1, 2.9, 4.7, 1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.6, 2.9, 3.6, 1.3</td>
<td>2.0, 6.7, 3.1, 4.4, 1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.6, 3.0, 4.5, 1.5</td>
<td>2.0, 5.8, 2.7, 4.1, 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.2, 2.2, 4.5, 1.5</td>
<td>2.0, 5.6, 2.5, 3.9, 1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.9, 3.2, 4.8, 1.8</td>
<td>2.0, 6.1, 2.8, 4.0, 1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.3, 2.5, 4.9, 1.5</td>
<td>2.0, 6.1, 2.8, 4.7, 1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.4, 2.9, 4.3, 1.3</td>
<td>2.0, 6.6, 3.0, 4.4, 1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.8, 2.8, 4.8, 1.4</td>
<td>2.0, 6.7, 3.0, 5.0, 1.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.0, 2.9, 4.5, 1.5</td>
<td>2.0, 5.7, 2.6, 3.5, 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.5, 2.4, 3.8, 1.1</td>
<td>2.0, 5.5, 2.4, 3.7, 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.8, 2.7, 3.9, 1.2</td>
<td>2.0, 6.0, 2.7, 5.1, 1.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.4, 3.0, 4.5, 1.5</td>
<td>2.0, 6.0, 3.4, 4.5, 1.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 6.7, 3.1, 4.7, 1.5</td>
<td>2.0, 6.3, 2.3, 4.4, 1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.6, 3.0, 4.1, 1.3</td>
<td>2.0, 5.5, 2.5, 4.0, 1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.5, 2.6, 4.4, 1.2</td>
<td>2.0, 6.1, 3.0, 4.6, 1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.8, 2.6, 4.0, 1.2</td>
<td>2.0, 5.0, 2.3, 3.3, 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.6, 2.7, 4.2, 1.3</td>
<td>2.0, 5.7, 3.0, 4.2, 1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.7, 2.9, 4.2, 1.3</td>
<td>2.0, 6.2, 2.9, 4.3, 1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0, 5.1, 2.5, 3.0, 1.1</td>
<td>2.0, 5.7, 2.8, 4.1, 1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 6.3, 3.3, 6.0, 2.5</td>
<td>3.0, 5.8, 2.7, 5.1, 1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 7.1, 3.0, 5.9, 2.1</td>
<td>3.0, 6.3, 2.9, 5.6, 1.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 6.5, 3.0, 5.8, 2.2</td>
<td>3.0, 7.6, 3.0, 6.6, 2.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 4.9, 2.5, 4.5, 1.7</td>
<td>3.0, 7.3, 2.9, 6.3, 1.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 6.7, 2.5, 5.8, 1.8</td>
<td>3.0, 7.2, 3.6, 6.1, 2.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 6.5, 3.2, 5.1, 2.0</td>
<td>3.0, 6.4, 2.7, 5.3, 1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 6.8, 3.0, 5.5, 2.1</td>
<td>3.0, 5.7, 2.5, 5.0, 2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 5.8, 2.8, 5.1, 2.4</td>
<td>3.0, 6.4, 3.2, 5.3, 2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 6.5, 3.0, 5.5, 1.8</td>
<td>3.0, 7.7, 3.8, 6.7, 2.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 7.7, 2.6, 6.9, 2.3</td>
<td>3.0, 6.0, 2.2, 5.0, 1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 6.9, 3.2, 5.7, 2.3</td>
<td>3.0, 5.6, 2.8, 4.9, 2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 7.7, 2.8, 6.7, 2.0</td>
<td>3.0, 6.3, 2.7, 4.9, 1.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0, 6.7, 3.3, 5.7, 2.1</td>
<td>3.0, 7.2, 3.2, 6.0, 1.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
int[] group = new int[x.GetLength(0)];
int[] varIndex = { 1, 2, 3, 4 };  
for (int i = 0; i < x.GetLength(0); i++)
{
    group[i] = (int)x[i,0];
}

int nvar = x.GetLength(1) - 1;
DiscriminantAnalysis da = new DiscriminantAnalysis(nvar, 3);
da.ClassificationMethod = DiscriminantAnalysis.Classification.Reclassification;
da.Update(x, group, varIndex);
da.Classify(x, group, varIndex);

PrintMatrixFormat pmf = new PrintMatrixFormat();
pmf.NumberFormat = "0.00";

new PrintMatrix("Xmean are: ").Print(pmf, da.GetMeans());
new PrintMatrix("Coeff: ").Print(pmf, da.GetCoefficients());
new PrintMatrix("Counts: ").Print(da.GetGroupCounts());
int[] cm = da.GetClassMembership();
int[,] cMem = new int[1, cm.Length];
for (int i = 0; i < cm.Length; i++)
{
    cMem[0, i] = cm[i];
}

new PrintMatrix("ClassMembership").SetPageWidth(50).Print(cMem);
new PrintMatrix("ClassTable: ").Print(da.GetClassTable());
double[,] tmpCov = new double[cov.GetLength(1), cov.GetLength(2)];
for (int i = 0; i < cov.GetLength(0); i++)
{
    for (int j = 0; j < cov.GetLength(1); j++)
    {
        for (int k = 0; k < cov.GetLength(2); k++)
        {
            tmpCov[j, k] = cov[i, j, k];
        }
    }
}
new PrintMatrix("Covariance Matrix " + i + ": ").Print(pmf, tmpCov);
new PrintMatrix("Prior : ").Print(da.GetPrior());
new PrintMatrix("PROB: ").Print(pmf, da.GetProbability());
new PrintMatrix("MAHALANOBIS: ").Print(pmf, da.GetMahalanobis());
Console.Out.WriteLine("nrmiss = " + da.NumberOfRowsMissing);
}

Output

Xmean are:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.01</td>
<td>3.43</td>
<td>1.46</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>5.94</td>
<td>2.77</td>
<td>4.26</td>
<td>1.33</td>
</tr>
<tr>
<td>2</td>
<td>6.59</td>
<td>2.97</td>
<td>5.55</td>
<td>2.03</td>
</tr>
</tbody>
</table>

Coeff:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-86.31</td>
<td>23.54</td>
<td>23.59</td>
<td>-16.43</td>
<td>-17.40</td>
</tr>
<tr>
<td>1</td>
<td>-72.85</td>
<td>15.70</td>
<td>7.07</td>
<td>5.21</td>
<td>6.43</td>
</tr>
<tr>
<td>2</td>
<td>-104.37</td>
<td>12.45</td>
<td>3.69</td>
<td>12.77</td>
<td>21.08</td>
</tr>
</tbody>
</table>

Counts:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
</tbody>
</table>

Stats:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>147.00</td>
</tr>
<tr>
<td>1</td>
<td>NaN</td>
</tr>
<tr>
<td>2</td>
<td>NaN</td>
</tr>
<tr>
<td>3</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>NaN</td>
</tr>
<tr>
<td>5</td>
<td>NaN</td>
</tr>
<tr>
<td>6</td>
<td>NaN</td>
</tr>
<tr>
<td>7</td>
<td>-9.96</td>
</tr>
<tr>
<td>8</td>
<td>50.00</td>
</tr>
<tr>
<td>9</td>
<td>50.00</td>
</tr>
<tr>
<td>10</td>
<td>50.00</td>
</tr>
<tr>
<td>11</td>
<td>150.00</td>
</tr>
</tbody>
</table>

ClassMembership

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>105</th>
<th>106</th>
<th>107</th>
<th>108</th>
<th>109</th>
<th>110</th>
<th>111</th>
<th>112</th>
<th>113</th>
<th>114</th>
<th>115</th>
<th>116</th>
<th>117</th>
<th>118</th>
<th>119</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
<td>128</td>
<td>129</td>
<td>130</td>
<td>131</td>
<td>132</td>
<td>133</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>135</td>
<td>136</td>
<td>137</td>
<td>138</td>
<td>139</td>
<td>140</td>
<td>141</td>
<td>142</td>
<td>143</td>
<td>144</td>
<td>145</td>
<td>146</td>
<td>147</td>
<td>148</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

ClassTable:
0 1 2
0  50 0 0
1  0 48 2
2  0  1  49

Covariance Matrix 0:
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.27</td>
<td>0.09</td>
<td>0.17</td>
<td>0.04</td>
</tr>
<tr>
<td>1</td>
<td>0.09</td>
<td>0.12</td>
<td>0.06</td>
<td>0.03</td>
</tr>
<tr>
<td>2</td>
<td>0.17</td>
<td>0.06</td>
<td>0.19</td>
<td>0.04</td>
</tr>
<tr>
<td>3</td>
<td>0.04</td>
<td>0.03</td>
<td>0.04</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Prior:
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0.333333333333333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0.333333333333333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0.333333333333333</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PROB:
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

Multivariate Analysis
Discriminant Analysis • 1057
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>17</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>18</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>19</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>20</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>21</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>22</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>23</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>24</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>25</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>26</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>27</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>28</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>29</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>30</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>31</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>32</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>33</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>34</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>35</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>36</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>37</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>38</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>39</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>40</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>41</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>42</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>43</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>44</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>45</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>46</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>47</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>48</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>49</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>50</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>51</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>52</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>53</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>54</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>55</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>56</td>
<td>0.00</td>
<td>0.99</td>
<td>0.01</td>
</tr>
<tr>
<td>57</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>58</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>59</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>60</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>61</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>62</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>63</td>
<td>0.00</td>
<td>0.99</td>
<td>0.01</td>
</tr>
<tr>
<td>64</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>65</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>66</td>
<td>0.00</td>
<td>0.98</td>
<td>0.02</td>
</tr>
<tr>
<td>67</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>68</td>
<td>0.00</td>
<td>0.96</td>
<td>0.04</td>
</tr>
<tr>
<td>69</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>70</td>
<td>0.00</td>
<td>0.25</td>
<td>0.75</td>
</tr>
<tr>
<td>71</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

1058 • DiscriminantAnalysis IMSL C# Numerical Library
Multivariate Analysis

Discriminant Analysis • 1059
DiscriminantAnalysis.Discrimination Enumeration


Discrimination methods.

Fields

Linear

public Imsl.Stat.DiscriminantAnalysis.Discrimination Linear

Description

Indicates a linear discrimination method.
**Quadratic**


**Description**
Indicates a quadratic discrimination method.

---

**DiscriminantAnalysis.CovarianceMatrix Enumeration**


Covariance matrix type.

**Fields**

**Pooled**


**Description**
Indicates pooled covariances computed.

**PooledGroup**


**Description**
Indicates pooled, group covariances computed.

---

**DiscriminantAnalysis.Classification Enumeration**

public enumeration Imsl.Stat.DiscriminantAnalysis.Classification

Classification method.
## Fields

**LeaveOutOne**

public Imsl.Stat.DiscriminantAnalysis.Classification LeaveOutOne

**Description**

Indicates leave-out-one as the classification method.

**Reclassification**

public Imsl.Stat.DiscriminantAnalysis.Classification Reclassification

**Description**

Indicates reclassification as the classification method.

---

## DiscriminantAnalysis.PriorProbabilities

**Enumeration**


Prior probabilities type.

---

## Fields

**Equal**


**Description**

Indicates prior probability type is to be prior equal.

**Proportional**


**Description**

Indicates prior probability type is to be prior proportional.
Chapter 20: Survival and Reliability Analysis

Types

class KaplanMeierECDF .............................................................. 1063
class KaplanMeierEstimates ...................................................... 1067
class ProportionalHazards .......................................................... 1075
enumeration ProportionalHazards.TieHandling ........................... 1092
class LifeTables ................................................................. 1092

Usage Notes

Survival Analysis

The functions described in this chapter have primary application in the areas of reliability and life testing, but they may find application in any situation in which analysis of binomial events over time is of interest. Kalbfleisch and Prentice (1980), Elandt-Johnson and Johnson (1980), Lee (1980), Gross and Clark (1975), Lawless (1982), and Chiang (1968) and Tanner and Wong (1984) are references for discussing the models and methods described in this chapter. Function KaplanMeierEstimates produces Kaplan-Meier (product-limit) estimates of the survival distribution in a single population. Function ProportionalHazards computes the parameter estimates in a proportional hazards model. Function SurvivalGLM fits any of several generalized linear models for survival data, and SurvivalEstimates computes estimates of survival probabilities based upon the same models. Function LifeTables computes and (optionally) prints an actuarial table based either upon a cohort followed over time or a cross-section of a population.

KaplanMeierECDF Class

public class Imsl.Stat.KaplanMeierECDF
Computes the Kaplan-Meier reliability function estimates or the CDF based on failure data that may be multi-censored.

The Kaplan-Meier (K-M) Product Limit procedure provides simple estimates of the reliability function or the CDF based on failure data that may be multi-censored. No underlying probability model is assumed; K-M estimation is an empirical (non-parametric) procedure. Exact times of failure are required.

Consider a situation in which we are reliability testing \( n \) (non-repairable) units taken randomly from a population. We are investigating the population to determine if its failure rate is acceptable. In the typical test scenario, we have a fixed time \( T \) to run the units to see if they survive or fail. The data obtained are called Censored Type I data.

During the \( T \) hours of test we observe \( r \) failures (where \( r \) can be any number from 0 to \( n \)). The failure times are \( t_1, t_2, \ldots, t_r \), and there are \( (n - r) \) units that survived the entire \( T \)-hour test without failing. Note that \( T \) is fixed in advance, and \( r \) is an output of the testing, since we don’t know how many failures will occur until the test is run. Note that we assume the exact times of failure are recorded when they occur.

This type of data is also called “right censored” data since the times of failure to the right (i.e., larger than \( T \)) are missing. The steps for calculating K-M estimates are the following:

1. Order the actual failure times from \( t_1 \) through \( t_r \), where there are \( r \) failures
2. Corresponding to each \( t_i \), associate the number \( n_i \) with \( n_i = \) the number of operating units just before the \( i \)th failure occurred at time \( t_i \)
3. First estimate the survival \( R(t_1) = (n_1 - 1)/n_1 \)
4. Estimate each ensuing survival \( R(t_i) = R(t_{i-1})(n_i - 1)/n_i, i > 1 \)
5. Estimate the CDF \( F(t_i) = 1 - R(t_i), i = 1, 2, \ldots \)

Note that non-failed units taken off testing (i.e., right-censored) only count up to the last actual failure time before they were removed. They are included in the \( n_i \) counts up to and including that failure time, but not after.

**Property**

*NumberOfPoints*

virtual public int NumberOfPoints {get; }

**Description**
The number of points in the empirical CDF.

**Property Value**
An int containing the number of points in the empirical CDF.

**Exception**
System.InvalidOperationException is thrown if the CDF has not been evaluated.
Constructor

KaplanMeierECDF
public KaplanMeierECDF(double[] t)

Description
Constructor for KaplanMeierECDF.

Parameter
t – A double array containing the failure times.

Methods

EvaluateCDF
virtual public double[] EvaluateCDF()

Description
Computes the empirical CDF and returns the CDF values up to, but not including the time values returned by GetTimes.

Returns
A double array of CDF values.

GetTimes
virtual public double[] GetTimes()

Description
Retrieves the time values where the step function CDF jumps to a greater value.

Returns
A double array of time values.

Remarks
The returned array has right-censored values of t removed.

Exception
System. InvalidOperationException is thrown if the CDF has not been evaluated.

SetCensor
virtual public void SetCensor(int[] censor)

Description
Set flags to note right-censoring.
Parameter
censor – An int array of 0 or 1 flags to note right-censoring. Values of 0 = continue to use datum
and 1 = remove datum.
Default: No data is right-censored.

SetFrequency
virtual public void SetFrequency(int[] freq)

Description
Sets the frequency for each entry in t.

Parameter
freq – An int array containing the repeat count for each entry in t.
Default: A frequency of 1 is used for each entry in t.

Example: Kaplan Meier Empirical CDF
This example illustrates the K-M procedure. Assume 20 units are on life test and 6 failures occur at
the following times: 10, 32, 56, 98, 122, and 181 hours. There were 4 working units removed from the test
for other experiments at the following times: 50, 100, 125, and 150 hours. The remaining 10 working
units were removed from the test at 200 hours. The K-M estimates for this life test are:

R(10) = 19/20
R(32) = 19/20 x 18/19
R(56) = 19/20 x 18/19 x 16/17
R(98) = 19/20 x 18/19 x 16/17 x 15/16
R(122) = 19/20 x 18/19 x 16/17 x 15/16 x 13/14
R(181) = 19/20 x 18/19 x 16/17 x 15/16 x 13/14 x 10/11

using System;
using Imsl.Stat;
using Imsl.Math;

public class KaplanMeierECDFEx1
{
    public static void Main(String[] args)
    {
        double[] y = {10.0, 32.0, 56.0, 98.0,
    122.0, 181.0, 50.0, 100.0,
    125.0, 150.0, 200.0};
        int[] freq = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 10};
        int[] censor = {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1};

        KaplanMeierECDF km = new KaplanMeierECDF(y);
        km.SetFrequency(freq);
        km.SetCensor(censor);
        double[] fx = km.EvaluateCDF();
    }
}
int ntimes = km.NumberOfPoints;
Console.Out.WriteLine("Number of points = " + ntimes);

double[] x = km.GetTimes();
PrintMatrix p =
new PrintMatrix("CDF = 1 - survival of life test subjects");
p.Print(fx);
p.SetTitle("Times of change in CDF");
p.Print(x);

Output

Number of points = 6
CDF = 1 - survival of life test subjects
  0 0.05
  1 0.1
  2 0.152941176470588
  3 0.205882352941177
  4 0.262605042016807
  5 0.329640947288006

Times of change in CDF
  0
  0 10
  1 32
  2 56
  3 98
  4 122
  5 181

KaplanMeierEstimates Class

public class Imsl.Stat.KaplanMeierEstimates

Computes Kaplan-Meier (or product-limit) estimates of survival probabilities for a sample of failure times that possibly contain right censoring.

Class KaplanMeierEstimates computes Kaplan-Meier (or product-limit) estimates of survival probabilities for a sample of failure times that can be right censored or exact times. A survival probability \( S(t) \) is defined as \( 1 - F(t) \), where \( F(t) \) is the cumulative distribution function of the failure times \( t \). Greenwood's estimate of the standard errors of the survival probability estimates are also computed. (See Kalbfleisch and Prentice, 1980, pages 13 and 14.)

Let \( (t_i, \delta_i) \), for \( i = 1, ..., n \) denote the failure censoring times and the censoring codes for the \( n \) observations in a single sample. Here, \( t_i = x_{i - 1, responseIndex} \) is a failure time if \( \delta_i = 0 \), where
\( \delta_i = x_i\text{-censorIndex}. \) Also, \( t_i \) is a right censoring time if \( \delta_i \) is 1. Rows in \( x \) containing values other than 0 or 1 for \( \delta_i \) are ignored. Let the number of observations in the sample that have not failed by time \( s_i \) be denoted by \( n_i \), where \( s_i \) is an ordered (from smallest to largest) listing of the distinct failure times (censoring times are omitted). Then the Kaplan-Meier estimate of the survival probabilities is a step function, which in the interval from \( s_i \) to \( s_{i+1} \) (including the lower endpoint) is given by

\[
\hat{S}(t) = \prod_{j=1}^{i} \left( \frac{n_j - d_j}{n_j} \right)
\]

where \( d_j \) denotes the number of failures occurring at time \( s_j \), and \( n_j \) is the number of observations that have not failed prior to \( s_j \).

Note that one row of \( x \) may correspond to more than one failed (or censored) observation when the frequency option is in effect (see \texttt{FrequencyColumn}). The Kaplan-Meier estimate of the survival probability prior to time \( s_1 \) is 1.0, while the Kaplan-Meier estimate of the survival probability after the last failure time is not defined.

Greenwood’s estimate of the variance of \( \hat{S}(t) \) in the interval from \( s_i \) to \( s_{i+1} \) is given as

\[
\text{est.var}(\hat{S}(t)) = \hat{S}^2(t) \sum_{j=1}^{i} \frac{d_j}{n_j(n_j - d_j)}
\]

KaplanMeierEstimates computes the single sample estimates of the survival probabilities for all samples of data included in \( x \) during a single call. This is accomplished through the stratum column of \( x \), which if present, must contain a distinct code for each sample of observations (see \texttt{StratumColumn}). If a stratum column is not specified, there is no grouping, and all observations are assumed to come from the same sample.

When failures and right-censored observations are tied and the data are to be sorted by KaplanMeierEstimates (\texttt{Sorted=true} is not used), KaplanMeierEstimates assumes that the time of censoring for the tied-censored observations is immediately after the tied failure (within the same sample). When \texttt{Sorted=true} is used, the data are assumed to be sorted from smallest to largest according to the response time column of \( x \) within each stratum (see \texttt{ResponseColumn}). Furthermore, a small increment of time is assumed (theoretically) to elapse between the failed and censored observations that are tied (in the same sample). Thus, when \texttt{Sorted=true} is used, the user must sort all of the data in \( x \) from smallest to largest according to the response time column (and the stratum column, if set). By appropriate sorting of the observations, the user can handle censored and failed observations that are tied in any manner desired.
Properties

CensorColumn
virtual public int CensorColumn {get; set; }

Description
The column index of x containing the optional censoring code for each observation.

Property Value
An int specifying the column index of x containing the optional censoring code for each observation.
Default: It is assumed that there is no censor code column in x. All observations are assumed to be exact failure times.

Remarks
If x[i,CensorColumn] equals 0, the failure time x[i,ResponseColumn] is treated as an exact time of failure. Otherwise, it is treated as right-censored time.

FrequencyColumn
virtual public int FrequencyColumn {get; set; }

Description
The column index of x containing the frequency of response for each observation.

Property Value
An int specifying the column index of x containing the frequency of response for each observation.
Default: It is assumed that there is no frequency response column recorded in x. Each observation in the data array is assumed to be for a single failure.

NumberOfRowsMissing
virtual public int NumberOfRowsMissing {get; }

Description
The number of rows of data in x that contain missing values in one or more specific columns of x.

Property Value
An int scalar representing the number of rows of data in x that contain missing values in one or more specific columns of x.

ResponseColumn
virtual public int ResponseColumn {get; set; }

Description
The column index of x containing the response time for each observation.

Property Value
An int specifying the column index of x containing the response time for each observation.
Default: ResponseColumn = 0.
Remarks
The interpretation of these times as either right-censored or exact failure times depends on the setting of the censor codes in the censor code column. See property CensorColumn.

Sorted

virtual public bool Sorted {get; set; }

Description
The boolean which indicates that the column of response times in x are already sorted.

Property Value
A boolean indicating whether or not column ResponseColumn of x is already sorted.
Default: It is assumed that column ResponseColumn of x is not sorted, so a detached sort is performed.

Remarks
Sorted = true indicates that column ResponseColumn of x is already sorted. Otherwise, a detached sort is performed prior to analysis. If sorting is performed, all censored individuals are assumed to follow tied failures.

StratumColumn

virtual public int StratumColumn {get; set; }

Description
The column index of x containing the stratum number for each observation.

Property Value
An int specifying the column index of x containing the stratum number for each observation.
Default: It is assumed that there is no stratum number column recorded in x. The data is assumed to come from one stratum.

Remarks
Column StratumColumn of x contains a unique value for each stratum in the data. Kaplan-Meier estimates are computed within each stratum.

Constructor

KaplanMeierEstimates

public KaplanMeierEstimates(double[,] x)

Description
Constructor for KaplanMeierEstimates.

Parameter
x – A double matrix containing the data, including optional data.
Default: It is assumed the response times are in column 0.
Methods

GetGroupTotal
virtual public int GetGroupTotal(double groupValue)

Description
Returns the total number in the group for the specified group value.

Parameter
    groupValue – A double specifying the group value.

Returns
An int representing the total number in the group which has value groupValue.

GetLogLikelihood
virtual public double GetLogLikelihood(double groupValue)

Description
Returns the Kaplan-Meier log-likelihood of the group with the specified group value.

Parameter
    groupValue – A double specifying the group value.

Returns
A double representing the Kaplan-Meier log-likelihood of the group which has value groupValue.

Remarks
The Kaplan-Meier log-likelihood is computed as:

$$\ell = \sum d_{(j)} \ln d_{(j)} + (n_{(j)} - d_{(j)}) \ln (n_{(j)} - d_{(j)}) - n_{(j)} \ln n_{(j)}$$

where the sum is with respect to the distinct failure times $s_{(j)}$.

GetNumberAtRisk
virtual public int[] GetNumberAtRisk()

Description
Returns the number of individuals at risk at each failure point.

Returns
An int array containing the number of individuals at risk at each failure point.

GetNumberOfFailures
virtual public int[] GetNumberOfFailures()

Description
Returns the number of failures which occurred at each failure point.
Returns
An int array containing the number of failures which occurred at each failure point.

GetStandardErrors
virtual public double[] GetStandardErrors()

Description
Returns Greenwood’s estimated standard errors.

Returns
A double array containing Greenwood’s estimate of the standard errors for the survival probabilities.

GetSurvivalProbabilities
public double[] GetSurvivalProbabilities()

Description
Returns the estimated survival probabilities.

Returns
A double array containing the estimated survival probabilities.

GetTotalNumberOfFailures
virtual public int GetTotalNumberOfFailures(double groupValue)

Description
Returns the total number failing in the group for the specified group value.

Parameter
    groupValue – A double specifying the group value.

Returns
An int representing the total number failing in the group which has value groupValue.

Example: KaplanMeierEstimates
The following example is taken from Kalbfleisch and Prentice (1980, page 1). The first column in x contains the death/censoring times for rats suffering from vaginal cancer. The second column contains information as to which of two forms of treatment were provided, while the third column contains the censoring code. Finally, the fourth column contains the frequency of each observation. The product-limit estimates of the survival probabilities are computed for both groups along with their standard error estimates. Tables containing these values along with other statistics are printed.

using System;
using Imsl.Stat;

public class KaplanMeierEstimatesEx1
{
    public static void Main(string[] args)
    {

1072 • KaplanMeierEstimates  IMSL C# Numerical Library
double[,] x = new double[,]{
    {143, 5, 0, 1}, {164, 5, 0, 1}, {188, 5, 0, 2}, {190, 5, 0, 1},
    {192, 5, 0, 1}, {206, 5, 0, 1}, {209, 5, 0, 1}, {213, 5, 0, 1},
    {216, 5, 0, 1}, {220, 5, 0, 1}, {227, 5, 0, 1}, {230, 5, 0, 1},
    {234, 5, 0, 1}, {246, 5, 0, 1}, {265, 5, 0, 1}, {284, 5, 0, 1},
    {216, 5, 1, 1}, {244, 5, 1, 1}, {142, 7, 0, 1}, {156, 7, 0, 1},
    {163, 7, 0, 1}, {198, 7, 0, 1}, {205, 7, 0, 1}, {232, 7, 0, 2},
    {233, 7, 0, 4}, {239, 7, 0, 1}, {248, 7, 0, 1}, {261, 7, 0, 1},
    {280, 7, 0, 2}, {296, 7, 0, 2}, {323, 7, 0, 1}, {204, 7, 1, 1},
    {344, 7, 1, 1}};
int nobs = x.GetLength(0), censorIndex = 2, frequencyIndex = 3;
int stratumIndex = 1, responseIndex = 0;
int i, groupValue;
int[] atRisk = new int[nobs];
int[] nFailing = new int[nobs];
double[] prob = new double[nobs];
double[] se = new double[nobs];

// Get Kaplan-Meier Estimates
KaplanMeierEstimates km = new KaplanMeierEstimates(x);
km.CensorColumn = censorIndex;
km.FrequencyColumn = frequencyIndex;
km.StratumColumn = stratumIndex;
atRisk = km.GetNumberAtRisk();
nFailing = km.GetNumberOfFailures();
prob = km.GetSurvivalProbabilities();
se = km.GetStandardErrors();

// Print Results
i = 0;
while (i < nobs)
{
    groupValue = (int) x[i, stratumIndex];
    Console.Out.WriteLine("\n Kaplan-Meier Survival "$ +
    "Probabilities");
    Console.Out.WriteLine(" For Group Value = " +
    "{0,5:0.####}", groupValue);
    Console.Out.WriteLine(" \nNumber Number" +
    "Survival Estimated");
    Console.Out.WriteLine("at risk Failing Time" +
    "Probability Std. Error");
    while (i < nobs && ((int) x[i, stratumIndex] == groupValue))
    {
        if ((int) x[i, censorIndex] != 1)
        {
            Console.Out.WriteLine("{0,5:0.####}" +
            "{1,5:0.####} {2,5:0.####} {3,5:0.####}" +
            "{4,5:0.####}" +
            "{5,5:0.####}" +
            "{6,5:0.####} atRisk[i], nFailing[i],
            (int) x[i, responseIndex], prob[i], se[i]);
        }
        i++;
    }
    Console.Out.WriteLine("\nTotal number in group = " +
    "{0,5:0.####}" +
    km.GetGroupTotal(groupValue);
    Console.Out.WriteLine("Total number failing = " +
    "{0,5:0.####}" +
    km.GetGroupTotal(groupValue - nFailing[i]) +
    "\nKaplanMeierEstimates • 1073"
Kaplan-Meier Survival Probabilities
For Group Value = 5

<table>
<thead>
<tr>
<th>Number at risk</th>
<th>Number Failing</th>
<th>Time</th>
<th>Survival Probability</th>
<th>Estimated Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>1</td>
<td>143</td>
<td>0.9474</td>
<td>0.0512</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>164</td>
<td>0.8947</td>
<td>0.0704</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>188</td>
<td>0.7895</td>
<td>0.0935</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>190</td>
<td>0.7368</td>
<td>0.101</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>192</td>
<td>0.6842</td>
<td>0.1066</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>206</td>
<td>0.6316</td>
<td>0.1107</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>209</td>
<td>0.5789</td>
<td>0.1133</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>213</td>
<td>0.5263</td>
<td>0.1145</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>216</td>
<td>0.4737</td>
<td>0.1145</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>220</td>
<td>0.4145</td>
<td>0.1145</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>227</td>
<td>0.3563</td>
<td>0.1124</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>230</td>
<td>0.2961</td>
<td>0.1082</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>234</td>
<td>0.2368</td>
<td>0.1015</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>246</td>
<td>0.1579</td>
<td>0.0934</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>265</td>
<td>0.0789</td>
<td>0.0728</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>304</td>
<td>0</td>
<td>NaN</td>
</tr>
</tbody>
</table>

Total number in group = 19
Total number failing = 17
Product Limit likelihood = -49.1692

Kaplan-Meier Survival Probabilities
For Group Value = 7

<table>
<thead>
<tr>
<th>Number at risk</th>
<th>Number Failing</th>
<th>Time</th>
<th>Survival Probability</th>
<th>Estimated Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>1</td>
<td>142</td>
<td>0.9524</td>
<td>0.0465</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>156</td>
<td>0.9048</td>
<td>0.0641</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>163</td>
<td>0.8571</td>
<td>0.0764</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>198</td>
<td>0.8095</td>
<td>0.0857</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>205</td>
<td>0.7589</td>
<td>0.0941</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>232</td>
<td>0.6577</td>
<td>0.1053</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>233</td>
<td>0.4564</td>
<td>0.1114</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>239</td>
<td>0.4048</td>
<td>0.1099</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>240</td>
<td>0.3542</td>
<td>0.1072</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>261</td>
<td>0.3036</td>
<td>0.1031</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>280</td>
<td>0.2024</td>
<td>0.0902</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>296</td>
<td>0.1012</td>
<td>0.0678</td>
</tr>
</tbody>
</table>
Total number in group = 21
Total number failing = 19
Product Limit likelihood = -50.4277

The number of rows of x with missing values is 0

---

**ProportionalHazards Class**

```java
public class Imsl.Stat.ProportionalHazards
```

Analyzes survival and reliability data using Cox’s proportional hazards model.

Class `ProportionalHazards` computes parameter estimates and other statistics in Proportional Hazards Generalized Linear Models. These models were first proposed by Cox (1972). Two methods for handling ties are allowed. Time-dependent covariates are not allowed. The user is referred to Cox and Oakes (1984), Kalbfleisch and Prentice (1980), Elandt-Johnson and Johnson (1980), Lee (1980), or Lawless (1982), among other texts, for a thorough discussion of the Cox proportional hazards model.

Let \( \lambda(t, z_i) \) represent the hazard rate at time \( t \) for observation number \( i \) with covariates contained as elements of row vector \( z_i \). The basic assumption in the proportional hazards model (the proportionality assumption) is that the hazard rate can be written as a product of a time varying function \( \lambda_0(t) \), which depends only on time, and a function \( f(z_i) \), which depends only on the covariable values. The function \( f(z_i) \) used in `ProportionalHazards` is given as \( f(z_i) = \exp(w_i + \beta z_i) \) where \( w_i \) is a fixed constant assigned to the observation, and \( \beta \) is a vector of coefficients to be estimated. With this function one obtains a hazard rate \( \lambda(t, z_i) = \lambda_0(t) \exp(w_i + \beta z_i) \). The form of \( \lambda_0(t) \) is not important in proportional hazards models.

The constants \( w_i \) may be known theoretically. For example, the hazard rate may be proportional to a known length or area, and the \( w_i \) can then be determined from this known length or area. Alternatively, the \( w_i \) may be used to fix a subset of the coefficients \( \beta \) (say, \( \beta_1 \)) at specified values. When \( w_i \) is used in this way, constants \( w_i = \beta_1 z_i \) are used, while the remaining coefficients in \( \beta \) are free to vary in the optimization algorithm. Constants are defined as 0.0 by default. If user-specified constants are desired, use the `ConstantColumn` property to specify which column contains the constant.

With this definition of \( \lambda(t, z_i) \), the usual partial (or marginal, see Kalbfleisch and Prentice (1980)) likelihood becomes

\[
L = \prod_{i=1}^{n_d} \frac{\exp(w_i + \beta z_i)}{\sum_{j \in R(t_i)} \exp(w_j + \beta z_j)}
\]

where \( R(t_i) \) denotes the set of indices of observations that have not yet failed at time \( t_i \) (the risk set), \( t_i \) denotes the time of failure for the \( i \)-th observation, \( n_d \) is the total number of observations that fail.
Right-censored observations (i.e., observations that are known to have survived to time $t_i$, but for which no time of failure is known) are incorporated into the likelihood through the risk set $R(t_i)$. Such observations never appear in the numerator of the likelihood. When TiesOption is set to BreslowsApproximate (the default), all observations that are censored at time $t_i$ are not included in $R(t_i)$, while all observations that fail at time $t_i$ are included in $R(t_i)$.

If it can be assumed that the dependence of the hazard rate upon the covariate values remains the same from stratum to stratum, while the time-dependent term, $\lambda_0(t)$, may be different in different strata, then ProportionalHazards allows the incorporation of strata into the likelihood as follows. Let $k$ index the $m$ strata (set with StratumColumn). Then, the likelihood is given by

$$L_S = \prod_{k=1}^{m} \left[ \prod_{i=1}^{n_k} \frac{\exp(w_{ki} + \beta z_{ki})}{\sum_{j \in R(t_{ki})} \exp(w_{kj} + \beta z_{kj})} \right]$$

In ProportionalHazards, the log of the likelihood is maximized with respect to the coefficients $\beta$. A quasi-Newton algorithm approximating the Hessian via the matrix of sums of squares and cross products of the first partial derivatives is used in the initial iterations. When the change in the log-likelihood from one iteration to the next is less than 100 times the convergence tolerance, Newton-Raphson iteration is used. If, during any iteration, the initial step does not lead to an increase in the log-likelihood, then step halving is employed to find a step that will increase the log-likelihood.

Once the maximum likelihood estimates have been computed, the algorithm computes estimates of a probability associated with each failure. Within stratum $k$, an estimate of the probability that the $i$-th observation fails at time $t_i$ given the risk set $R(t_{ki})$ is given by

$$p_{ki} = \frac{\exp(w_{ki} + \beta z_{ki})}{\sum_{j \in R(t_{ki})} \exp(w_{kj} + \beta z_{kj})}$$

A diagnostic “influence” or “leverage” statistic is computed for each noncensored observation as:

$$l_{ki} = -g'_{ki} H^{-1} g'_{ki}$$

where $H_s$ is the matrix of second partial derivatives of the log-likelihood, and

$$g'_{ki}$$

is computed as:

$$g'_{ki} = z_{ki} - \frac{z_{ki} \exp(w_{ki} + \beta z_{ki})}{\sum_{j \in R(t_{ki})} \exp(w_{kj} + \beta z_{kj})}$$

Influence statistics are not computed for censored observations.

A “residual” is computed for each of the input observations according to methods given in Cox and Oakes (1984, page 108). Residuals are computed as

1076 • ProportionalHazards  IMSL C# Numerical Library
\[ r_{ki} = \exp(w_{ki} + \hat{\beta}z_{ki}) \sum_{j \in R(k_i)} \frac{d_{kj}}{\sum_{l \in R(t_{kj})} \exp(w_{kl} + \hat{\beta}z_{kl})} \]

where \( d_{kj} \) is the number of tied failures in group \( k \) at time \( t_{kj} \). Assuming that the proportional hazards assumption holds, the residuals should approximate a random sample (with censoring) from the unit exponential distribution. By subtracting the expected values, centered residuals can be obtained. (The \( j \)-th expected order statistic from the unit exponential with censoring is given as

\[ e_j = \sum_{l \leq j} \frac{1}{h - l + 1} \]

where \( h \) is the sample size, and censored observations are not included in the summation.)

An estimate of the cumulative baseline hazard within group \( k \) is given as

\[ \hat{H}_{k0}(t_{ik}) = \sum_{t_{kj} \leq t_{ki}, k \in R(t_{kj})} \frac{d_{kj}}{\sum_{l \in R(t_{kj})} \exp(w_{kl} + \hat{\beta}z_{kl})} \]

The observation proportionality constant is computed as

\[ \exp(w_{ki} + \hat{\beta}z_{ki}) \]

Note that one can use logging to generate intermediate output for this class. Accumulated levels of detail correspond to Fine, Finer, and Finest logging levels with Fine yielding the smallest amount of information and Finest yielding the most. The levels of output yield the following:

<table>
<thead>
<tr>
<th>Level</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine</td>
<td>Logging is enabled, but observational statistics are not printed.</td>
</tr>
<tr>
<td>Finer</td>
<td>All output statistics are printed.</td>
</tr>
<tr>
<td>Finest</td>
<td>Tracks progress through internal methods.</td>
</tr>
</tbody>
</table>

**Properties**

**CensorColumn**

virtual public int CensorColumn {get; set; }

**Description**

The column index of \( x \) containing the optional censoring code for each observation.
**Property Value**
An int specifying the column index of x containing the optional censoring code for each observation. Default: It is assumed that there is no censor code column in x and all observations are assumed to be exact failure times.

**Remarks**
If x[i][CensorColumn] equals 0, the failure time x[i][CensorColumn] is treated as an exact time of failure. Otherwise, it is treated as right-censored time.

---

**ConstantColumn**
virtual public int ConstantColumn {get; set; }

**Description**
The column index of x containing the constant to be added to the linear response.

**Property Value**
An int specifying the column index of x containing the constant to be added to the linear response. Default: It is assumed that \( w_i = 0 \) for all observations.

**Remarks**
The linear response is taken to be \( w_i + z_i \hat{\beta} \) where \( w_i \) is the observation constant, \( z_i \) is the observation design row vector, and \( \hat{\beta} \) is the vector of estimated parameters. The “fixed” constant allows one to test hypotheses about parameters via the log-likelihoods.

---

**ConvergenceTol**
virtual public double ConvergenceTol {get; set; }

**Description**
The convergence tolerance.

**Property Value**
a double specifying the convergence tolerance. Default: ConvergenceTol is 0.0001.

**Remarks**
Convergence is assumed when the relative change in the maximum likelihood from one iteration to the next is less than ConvergenceTol. If ConvergenceTol is zero, ConvergenceTol = 0.0001 is assumed.

---

**FrequencyColumn**
virtual public int FrequencyColumn {get; set; }

**Description**
The column index of x containing the frequency of response for each observation.
Property Value
An int specifying the column index of x containing the frequency of response for each observation.
Default: It is assumed that there is no frequency response column recorded in x. Each observation in the data array is assumed to be for a single failure; that is, the frequency of response for each observation is 1.

**HessianOption**
virtual public bool HessianOption {get; set; }

Description
The boolean used to indicate whether or not to compute the Hessian and gradient at the initial estimates.

Property Value
A boolean specifying whether or not the Hessian and gradient are to be computed at the initial estimates.
Default: The Hessian and gradient are not computed at the initial estimates.

Remarks
HessianOption equal to true indicates that the Hessian and gradient are to be computed. If HessianOption is true the user must set the initial estimates via the SetInitialEstimates method.

**Logger**
virtual public Imsl.Logger Logger {get; set; }

Description
Returns the logger object and enables logging.

Property Value
A Imsl.Logger object, if present, or null.

**MaxClass**
virtual public int MaxClass {get; set; }

Description
The upper bound used on the sum of the number of distinct values found among the classification variables in x.

Property Value
An int representing the upper bound used on the sum of the number of distinct values found among the classification variables in x.
Default: MaxClass is the number of observations in x.

Remarks
Consider a model consisting of two class variables, one with the values \{1, 2, 3, 4\} and a second with the values \{0, 1\}, then the total number of different classification values is 4 + 2 = 6, and MaxClass \(\geq 6\).
Description
Returns the maximized log-likelihood.

Property Value
A double representing the maximized log-likelihood.

Remarks
The log-likelihood is fully described in the ProportionalHazards class description.

Exception
Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through MaxClass has been exceeded.

MaxIterations
virtual public int MaxIterations {get; set; }

Description
The maximum number of iterations allowed.

Property Value
An int specifying the maximum number of iterations allowed.
Default: MaxIterations is 30.

NumberOfCoefficients
virtual public int NumberOfCoefficients {get; }

Description
Returns the number of estimated coefficients in the model.

Property Value
An int scalar representing the number of estimated coefficients in the model.

Exception
Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

NumberOfRowsMissing
virtual public int NumberOfRowsMissing {get; }

Description
The number of rows of data in x that contain missing values in one or more columns of x.

Property Value
An int scalar representing the number of rows of data in x that contain missing values in one or more columns of x.
Exception

Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

**ResponseColumn**

```csharp
virtual public int ResponseColumn {get; set; }
```

**Description**
The column index of \( x \) containing the response time for each observation.

**Property Value**
An int specifying the column index of \( x \) containing the response time for each observation.

Default: ResponseColumn = 0.

**Remarks**
For point observations, \( x[i][\text{ResponseColumn}] \) contains the time of the \( i \)-th event. For right-censored observations, \( x[i][\text{ResponseColumn}] \) contains the right-censoring time. Note that because ProportionalHazards only uses the order of the events, negative “times” are allowed.

**StratumColumn**

```csharp
virtual public int StratumColumn {get; set; }
```

**Description**
The column index of \( x \) containing the stratum number for each observation.

**Property Value**
An int specifying the column index of \( x \) containing the stratum number for each observation.

Default: It is assumed that all observations are from one stratum.

**Remarks**
Column StratumColumn of \( x \) contains a unique value for each stratum in the data. The risk set for an observation is determined by its stratum.

**StratumRatio**

```csharp
virtual public double StratumRatio {get; set; }
```

**Description**
The ratio at which a stratum is split into two strata.

**Property Value**
A double specifying the ratio at which a stratum is split into two strata.

Default: StratumRatio = 1000.
Remarks
Let
\[ r_k = \exp(z_k \beta + w_k) \]
be the observation proportionality constant, where \( z_k \) is the design row vector for the \( k \)-th observation and \( w_k \) is the optional fixed parameter specified by \( x_k \), \( \text{ConstantColumn} \). Let \( r_{min} \) be the minimum value \( r_k \) in a stratum, where, for failed observations, the minimum is over all times less than or equal to the time of occurrence of the \( k \)-th observation. Let \( r_{max} \) be the maximum value of \( r_k \) for the remaining observations in the group. Then, if \( r_{min} > \text{value} \times r_{max} \), the observations in the group are divided into two groups at \( k \). Set \( \text{StratumRatio} = -1 \) if no division into strata is to be made.

TiesOption
virtual public Imsl.Stat.ProportionalHazards.TieHandling TiesOption {get; set; }

Description
The method used for handling ties.

Property Value
A ProportionalHazards.TieHandling specifying the method to be used in handling ties.

Remarks
TiesOption is one of the values in the following table:

<table>
<thead>
<tr>
<th>( \text{value} )</th>
<th>( \text{Method} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{BreslowsApproximate}</td>
<td>Breslow’s approximate method. This is the default.</td>
</tr>
<tr>
<td>\text{SortedAsPerObservations}</td>
<td>Failures are assumed to occur in the same order as the observations input in ( x ). The observations in ( x ) must be sorted from largest to smallest failure time within each stratum, and grouped by stratum. All observations are treated as if their failure/censoring times were distinct when computing the log-likelihood.</td>
</tr>
</tbody>
</table>

Constructor

ProportionalHazards
class ProportionalHazards

public ProportionalHazards(double[,] x, int[] nVarEffects, int[] indEffects)

Description
Constructor for ProportionalHazards.
Parameters

- **x** – A double matrix containing the data, including optional data.
- **nVarEffects** – An int array containing the number of variables associated with each effect in the model.
- **indEffects** – An int array containing the column numbers of *x* associated with each effect. The first *nVarEffects[0]* elements of **indEffects** contain the column numbers of *x* for the variables in the first effect. The next *nVarEffects[1]* elements of **indEffects** contain the column numbers of *x* for the variables in the second effect, etc.

Methods

**GetCaseStatistics**

```csharp
virtual public double[,] GetCaseStatistics()
```

Description

Returns the case statistics for each observation.

Returns

A double matrix containing the case statistics.

Remarks

There is one row for each observation, and the columns of the returned matrix contain the following:

<table>
<thead>
<tr>
<th>Column</th>
<th>Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Estimated survival probability at the observation time.</td>
</tr>
<tr>
<td>1</td>
<td>Estimated observation influence or leverage.</td>
</tr>
<tr>
<td>2</td>
<td>A residual estimate.</td>
</tr>
<tr>
<td>3</td>
<td>Estimated cumulative baseline hazard rate.</td>
</tr>
<tr>
<td>4</td>
<td>Observation proportionality constant.</td>
</tr>
</tbody>
</table>

Exception

**Imsl.Stat.ClassificationVariableLimitException** is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

**GetClassValueCounts**

```csharp
virtual public int[] GetClassValueCounts()
```

Description

Returns the number of values taken by each classification variable.

Returns

An int array containing the number of values taken by each classification variable.
Remarks
The \( i \)-th element of the returned array is the number of distinct values taken by the \( i \)-th classification variable.

Exception
Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

GetClassValues
virtual public double[] GetClassValues()

Description
Returns the class values taken by each classification variable.

Returns
A double array containing the values taken by each classification variable.

Remarks
For description purposes, let \( nclval = \text{GetClassValueCounts}() \). Then the first \( nclval[0] \) elements contain the values for the first classification variable, the next \( nclval[1] \) elements contain the values for the second classification variable, etc.

Exception
Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

GetGradient
virtual public double[] GetGradient()

Description
Returns the inverse of the Hessian times the gradient vector, computed at the initial estimates.

Returns
A double array containing the inverse of the Hessian times the gradient vector, computed at the initial estimates.

Remarks
Note that the HessianOption property must be set to true and the SetInitialEstimates method must be invoked prior to invoking this method. Otherwise, the method throws an InvalidOperationException exception.

Exception
Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

GetHessian
virtual public double[,] GetHessian()
Description
Returns the inverse of the Hessian of the negative of the log-likelihood, computed at the initial estimates.

Returns
A double matrix containing the inverse of the Hessian of the negative of the log-likelihood, computed at the initial estimates.

Remarks
Note that the HessianOption property must set to true and the SetInitialEstimates method must be invoked prior to invoking this method. Otherwise, the method throws an InvalidOperationException exception.

Exception
Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

GetInitialEstimates
virtual public double[] GetInitialEstimates()

Description
Gets the initial parameter estimates.

Returns
A double array containing the initial parameter estimates.

GetLastUpdates
virtual public double[] GetLastUpdates()

Description
Gets the last parameter updates.

Returns
A double array containing the last parameter updates (excluding step halvings).

Exception
Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

GetMeans
virtual public double[] GetMeans()

Description
Returns the means of the design variables.

Returns
A double array containing the means of the design variables.
Exception

Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

GetParameterStatistics
virtual public double[,] GetParameterStatistics()

Description
Returns the parameter estimates and associated statistics.

Returns
A double matrix containing the parameter estimates and associated statistics.

Remarks
There is one row for each coefficient, and the columns of the returned matrix contain the following:

<table>
<thead>
<tr>
<th>Column</th>
<th>Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The coefficient estimate, ( \hat{\beta} )</td>
</tr>
<tr>
<td>1</td>
<td>Estimated standard deviation of the estimated coefficient</td>
</tr>
<tr>
<td>2</td>
<td>Asymptotic normal score for testing that the coefficient is zero against the two-sided alternative</td>
</tr>
<tr>
<td>3</td>
<td>( p )-value associated with the normal score in column 2</td>
</tr>
</tbody>
</table>

Exception

Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

GetStratumNumbers
virtual public int[] GetStratumNumbers()

Description
Returns the stratum number used for each observation.

Returns
An int array containing the stratum number used for each observation.

Remarks
If property StratumRatio is not -1.0, additional “strata” (other than those specified by the column of \( x \) set via the StratumColumn property) may be generated. The array also contains a record of the generated strata. See the ProportionalHazards class description for more detail.

Exception

Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

GetVarianceCovarianceMatrix
virtual public double[,] GetVarianceCovarianceMatrix()
Description
Returns the estimated asymptotic variance-covariance matrix of the parameters.

Returns
A double matrix containing the estimated asymptotic variance-covariance matrix of the parameters.

Exception
Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

SetClassVarColumns
virtual public void SetClassVarColumns(int[] classVarIndices)

Description
Sets the column indices of x that are the classification variables.

Parameter
classVarIndices – An int array containing the column numbers of x that are the classification variables.

Remarks
Default: It is assumed there are no classification variables.

SetInitialEstimates
virtual public void SetInitialEstimates(double[] initialCoef)

Description
Sets the initial parameter estimates.

Parameter
initialCoef – A double array containing the initial parameter estimates.

Remarks
Care should be taken to ensure that the supplied estimates for the model coefficients $\beta$ correspond to the generated covariate vector $z_{ki}$.
Default: The initial parameter estimates are all 0.0.

Exception
Imsl.Stat.ClassificationVariableLimitException is thrown if the classification variable limit set by the user through the MaxClass property has been exceeded.

Example: ProportionalHazards
The following example is taken from Lawless (1982, page 287) and involves the survival of lung cancer patients based upon their initial tumor types and treatment type. In the example, the likelihood is maximized with no strata present in the data. This corresponds to Example 7.2.3 in Lawless (1982, page 367). The model is given as:
\[ \ln(\lambda) = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \alpha_i + \gamma_j \]

where \( \alpha_i \) and \( \gamma_j \) correspond to dummy variables generated from classification variables in columns 5 and 6 of \( x \). Respectively, \( x_1 \) corresponds to column index 2, \( x_2 \) corresponds to column index 3, and \( x_3 \) corresponds to column index 4 of \( x \). Column 0 of \( x \) contains the response and column 1 of \( x \) contains the censoring code. Logging is used to print output statistics.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl;

public class ProportionalHazardsEx1
{
    public static void Main(string[] args)
    {
        double[,] x =
            {{411, 0, 7, 64, 5, 1, 0}, {126, 0, 6, 63, 9, 1, 0},
             {118, 0, 7, 65, 11, 1, 0}, {92, 0, 4, 69, 10, 1, 0},
             {8, 0, 4, 63, 58, 1, 0}, {25, 1, 7, 48, 9, 1, 0},
             {11, 0, 7, 48, 11, 1, 0}, {54, 0, 8, 63, 4, 2, 0},
             {153, 0, 6, 63, 14, 2, 0}, {16, 0, 3, 53, 4, 2, 0},
             {56, 0, 8, 43, 12, 2, 0}, {21, 0, 4, 55, 2, 2, 0},
             {287, 0, 6, 66, 25, 2, 0}, {10, 0, 4, 67, 23, 2, 0},
             {8, 0, 2, 61, 19, 3, 0}, {12, 0, 5, 63, 4, 3, 0},
             {177, 0, 5, 66, 16, 4, 0}, {12, 0, 4, 68, 12, 4, 0},
             {200, 0, 8, 41, 12, 4, 0}, {250, 0, 7, 53, 8, 4, 0},
             {100, 0, 6, 37, 13, 4, 0}, {999, 0, 9, 54, 12, 1, 1},
             {231, 1, 5, 52, 8, 1, 1}, {991, 0, 7, 50, 7, 1, 1},
             {1, 0, 2, 65, 21, 1, 1}, {201, 0, 8, 52, 28, 1, 1},
             {44, 0, 6, 70, 13, 1, 1}, {15, 0, 5, 40, 13, 1, 1},
             {103, 1, 7, 36, 22, 2, 1}, {2, 0, 4, 44, 36, 2, 1},
             {20, 0, 3, 54, 9, 2, 1}, {51, 0, 3, 59, 87, 2, 1},
             {18, 0, 4, 69, 5, 3, 1}, {90, 0, 6, 50, 22, 3, 1},
             {84, 0, 8, 62, 4, 3, 1}, {164, 0, 7, 68, 15, 4, 1},
             {19, 0, 3, 39, 4, 4, 1}, {43, 0, 6, 49, 11, 4, 1},
             {340, 0, 8, 64, 10, 4, 1}, {231, 0, 7, 67, 18, 4, 1}};
        int[] indef = {2, 3, 4, 5, 6};
        int[] nvef = {1, 1, 1, 1, 1};
        int[] indcl = {5, 6};
        int maxcl = 6, icen = 1;
        double ratio = 10000.0;

        ProportionalHazards ph = new ProportionalHazards(x, nvef, indef);
        ph.MaxClass = maxcl;
        ph.CensorColumn = icen;
        ph.SetClassVarColumns(indcl);
        ph.StratumRatio = ratio;

        // Level.FINER prints most output statistics

        Logger logger = ph.Logger;
        logger.LogLevel = Logger.Level.Finer;
    }
}
double[,] coef = ph.GetParameterStatistics();
new PrintMatrix("\nFinal Coefficient Matrix").Print(coef);
}

### Output

**ProportionalHazards: Initial Estimates**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**ProportionalHazards: Method Iteration Step Maximum scaled Log**

<table>
<thead>
<tr>
<th>ProportionalHazards:</th>
<th>Size</th>
<th>coef. update</th>
<th>likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-N</td>
<td>0</td>
<td>-102.400565515673</td>
<td></td>
</tr>
<tr>
<td>Q-N</td>
<td>1</td>
<td>1 0.50338404715466</td>
<td>-91.0439507780637</td>
</tr>
<tr>
<td>Q-N</td>
<td>2</td>
<td>1 0.578199657315753</td>
<td>-88.0680849909176</td>
</tr>
<tr>
<td>N-R</td>
<td>3</td>
<td>1 0.113104528935929</td>
<td>-87.9223344687152</td>
</tr>
<tr>
<td>N-R</td>
<td>4</td>
<td>1 0.069579511007414</td>
<td>-87.887787543875</td>
</tr>
<tr>
<td>N-R</td>
<td>5</td>
<td>1 0.00081490486501336</td>
<td>-87.8877800993911</td>
</tr>
</tbody>
</table>

ProportionalHazards: Log-Likelihood = -87.8877800993911

**ProportionalHazards: Coefficient Statistics**

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Std. error</th>
<th>Asymptotic z-stat</th>
<th>Asymptotic p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.5846</td>
<td>0.1368</td>
<td>-4.2721</td>
</tr>
<tr>
<td>1</td>
<td>-0.0131</td>
<td>0.0206</td>
<td>-0.6342</td>
</tr>
<tr>
<td>2</td>
<td>0.0008</td>
<td>0.0118</td>
<td>0.0645</td>
</tr>
<tr>
<td>3</td>
<td>-0.367</td>
<td>0.4848</td>
<td>-0.7572</td>
</tr>
<tr>
<td>4</td>
<td>-0.0077</td>
<td>0.5068</td>
<td>-0.0152</td>
</tr>
<tr>
<td>5</td>
<td>1.1129</td>
<td>0.6331</td>
<td>1.758</td>
</tr>
<tr>
<td>6</td>
<td>0.3797</td>
<td>0.4058</td>
<td>0.9357</td>
</tr>
</tbody>
</table>

**ProportionalHazards: Case Analysis**

<table>
<thead>
<tr>
<th>Survival Prob.</th>
<th>Influence</th>
<th>Residual</th>
<th>Cumulative hazard</th>
<th>Prop. constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0022</td>
<td>0.0414</td>
<td>2.0531</td>
<td>6.1032</td>
</tr>
<tr>
<td>1</td>
<td>0.2988</td>
<td>0.1088</td>
<td>0.7409</td>
<td>1.2078</td>
</tr>
<tr>
<td>2</td>
<td>0.3424</td>
<td>0.1184</td>
<td>0.3576</td>
<td>1.0719</td>
</tr>
<tr>
<td>3</td>
<td>0.4336</td>
<td>0.1554</td>
<td>1.5272</td>
<td>0.8357</td>
</tr>
</tbody>
</table>

---

**Survival and Reliability Analysis**

**ProportionalHazards ● 1089**
<table>
<thead>
<tr>
<th></th>
<th>0.9555</th>
<th>0.5567</th>
<th>0.0933</th>
<th>0.0455</th>
<th>2.0499</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.7365</td>
<td>NaN</td>
<td>0.1272</td>
<td>0.3058</td>
<td>0.4158</td>
</tr>
<tr>
<td>5</td>
<td>0.9204</td>
<td>0.3729</td>
<td>0.0346</td>
<td>0.083</td>
<td>0.4164</td>
</tr>
<tr>
<td>6</td>
<td>0.5876</td>
<td>0.2637</td>
<td>0.1446</td>
<td>0.5317</td>
<td>0.2719</td>
</tr>
<tr>
<td>7</td>
<td>0.2577</td>
<td>0.1173</td>
<td>1.196</td>
<td>1.3561</td>
<td>0.882</td>
</tr>
<tr>
<td>8</td>
<td>0.8457</td>
<td>0.1486</td>
<td>0.9656</td>
<td>0.1676</td>
<td>5.7608</td>
</tr>
<tr>
<td>9</td>
<td>0.5481</td>
<td>0.3133</td>
<td>0.2135</td>
<td>0.6012</td>
<td>0.3551</td>
</tr>
<tr>
<td>10</td>
<td>0.7365</td>
<td>0.2108</td>
<td>0.9551</td>
<td>0.3058</td>
<td>3.1232</td>
</tr>
<tr>
<td>11</td>
<td>0.0293</td>
<td>0.0602</td>
<td>3.018</td>
<td>3.5289</td>
<td>0.8552</td>
</tr>
<tr>
<td>12</td>
<td>0.9382</td>
<td>0.0935</td>
<td>0.173</td>
<td>0.0638</td>
<td>2.7135</td>
</tr>
<tr>
<td>13</td>
<td>0.9555</td>
<td>0.1595</td>
<td>1.3142</td>
<td>0.0455</td>
<td>28.8855</td>
</tr>
<tr>
<td>14</td>
<td>0.8854</td>
<td>0.2322</td>
<td>0.5864</td>
<td>0.1217</td>
<td>4.8164</td>
</tr>
<tr>
<td>15</td>
<td>0.1814</td>
<td>0.0918</td>
<td>2.6217</td>
<td>1.707</td>
<td>1.5358</td>
</tr>
<tr>
<td>16</td>
<td>0.8854</td>
<td>0.1869</td>
<td>0.3258</td>
<td>0.1217</td>
<td>2.6765</td>
</tr>
<tr>
<td>17</td>
<td>0.1414</td>
<td>0.2303</td>
<td>0.7187</td>
<td>1.9565</td>
<td>0.3673</td>
</tr>
<tr>
<td>18</td>
<td>0.0522</td>
<td>0.0943</td>
<td>1.6591</td>
<td>2.9529</td>
<td>0.5618</td>
</tr>
<tr>
<td>19</td>
<td>0.3899</td>
<td>0.2212</td>
<td>1.1745</td>
<td>0.9419</td>
<td>1.2469</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>0</td>
<td>1.7281</td>
<td>21.1049</td>
<td>0.0819</td>
</tr>
<tr>
<td>21</td>
<td>0.0806</td>
<td>NaN</td>
<td>2.1865</td>
<td>2.5177</td>
<td>0.8684</td>
</tr>
<tr>
<td>22</td>
<td>0.0001</td>
<td>0.0049</td>
<td>2.4603</td>
<td>8.8921</td>
<td>0.2767</td>
</tr>
<tr>
<td>23</td>
<td>0.9892</td>
<td>0.3072</td>
<td>0.0462</td>
<td>0.0108</td>
<td>4.2758</td>
</tr>
<tr>
<td>24</td>
<td>0.1074</td>
<td>0.1724</td>
<td>0.3406</td>
<td>2.2311</td>
<td>0.1527</td>
</tr>
<tr>
<td>25</td>
<td>0.664</td>
<td>0.2513</td>
<td>0.1573</td>
<td>0.4095</td>
<td>0.3841</td>
</tr>
<tr>
<td>26</td>
<td>0.8655</td>
<td>0.2215</td>
<td>0.1472</td>
<td>0.1444</td>
<td>1.0196</td>
</tr>
<tr>
<td>27</td>
<td>0.3899</td>
<td>NaN</td>
<td>0.4533</td>
<td>0.9419</td>
<td>0.4812</td>
</tr>
<tr>
<td>28</td>
<td>0.9781</td>
<td>0.2495</td>
<td>0.0561</td>
<td>0.0222</td>
<td>2.531</td>
</tr>
<tr>
<td>29</td>
<td>0.769</td>
<td>0.2556</td>
<td>1.0257</td>
<td>0.2627</td>
<td>3.9045</td>
</tr>
<tr>
<td>30</td>
<td>0.6291</td>
<td>0.3509</td>
<td>1.7991</td>
<td>0.4635</td>
<td>3.8817</td>
</tr>
<tr>
<td>31</td>
<td>0.8233</td>
<td>0.2598</td>
<td>1.0635</td>
<td>0.1944</td>
<td>5.4705</td>
</tr>
<tr>
<td>32</td>
<td>0.4739</td>
<td>0.26</td>
<td>1.6474</td>
<td>0.7468</td>
<td>2.2058</td>
</tr>
<tr>
<td>33</td>
<td>0.5104</td>
<td>0.3191</td>
<td>0.3886</td>
<td>0.6725</td>
<td>0.5779</td>
</tr>
<tr>
<td>34</td>
<td>0.2173</td>
<td>0.183</td>
<td>0.485</td>
<td>1.5267</td>
<td>0.3177</td>
</tr>
<tr>
<td>35</td>
<td>0.7979</td>
<td>0.2642</td>
<td>1.0764</td>
<td>0.2258</td>
<td>4.7676</td>
</tr>
<tr>
<td>36</td>
<td>0.7</td>
<td>0.16</td>
<td>0.2598</td>
<td>0.3567</td>
<td>0.7282</td>
</tr>
<tr>
<td>37</td>
<td>0.0094</td>
<td>0.2267</td>
<td>0.8668</td>
<td>4.6642</td>
<td>0.1858</td>
</tr>
<tr>
<td>38</td>
<td>0.0806</td>
<td>0.2045</td>
<td>0.8122</td>
<td>2.5177</td>
<td>0.3226</td>
</tr>
</tbody>
</table>

ProportionalHazards: Last Coefficient Update

|   | -5.835E-8 | 1.401E-9 | -8.597E-9 | -2.822E-7 | -4.566E-8 | 1.256E-7 | 1.058E-8 |

ProportionalHazards: Covariate Means

|   | 5.65 | 56.575 | 15.65 | 0.35 | 0.275 |
5  0.125
6  0.525

ProportionalHazards: Distinct Values For Each Class Variable
ProportionalHazards: Variable 0: 1 2 3 4
ProportionalHazards: Variable 1: 0 1
ProportionalHazards: Stratum Numbers For Each Observation

0 1
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1
11 1
12 1
13 1
14 1
15 1
16 1
17 1
18 1
19 1
20 1
21 1
22 1
23 1
24 1
25 1
26 1
27 1
28 1
29 1
30 1
31 1
32 1
33 1
34 1
35 1
36 1
37 1
38 1
39 1

ProportionalHazards: Number of Missing Values = 0

Final Coefficient Matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.584594437935942</td>
<td>0.136840243992857</td>
<td>-4.27209438450326</td>
<td>1.93645575283785E-05</td>
</tr>
</tbody>
</table>

Survival and Reliability Analysis

ProportionalHazards ● 1091
ProportionalHazards.TieHandling Enumeration

public enumeration Imsl.Stat.ProportionalHazards.TieHandling

Tie handling options.

Fields

BreslowsApproximate
public Imsl.Stat.ProportionalHazards.TieHandling BreslowsApproximate

Description
Indicates Breslows approximate method of handling ties.

SortedAsPerObservations
public Imsl.Stat.ProportionalHazards.TieHandling SortedAsPerObservations

Description
Indicates failures are assumed to occur in the same order as the observations input in x.

Remarks
The observations in x must be sorted from largest to smallest failure time within each stratum, and grouped by stratum. All observations are treated as if their failure/censoring times were distinct when computing the log-likelihood.

LifeTables Class

public class Imsl.Stat.LifeTables
Computes population (current) or cohort life tables based upon the observed population sizes at the middle (for population table) or the beginning (for cohort table) of some user specified age intervals. The number of deaths in each of these intervals must also be observed.

The probability of dying prior to the middle of the interval, given that death occurs somewhere in the interval, may also be specified. Often, however, this probability is taken to be 0.5. For a discussion of the probability models underlying the life table here, see the references.

Let $t_i$, for $i = 0, 1, \ldots, t_n$ denote the time grid defining the $n$ age intervals, and note that the length of the age intervals may vary. Following Gross and Clark (1975, page 24), let $d_i$ denote the number of individuals dying in age interval $i$, where age interval $i$ ends at time $t_i$. For population table, the death rate at the middle of the interval is given by $r_i = d_i/(M_i h_i)$, where $M_i$ is the number of individuals alive at the middle of the interval, and $h_i = t_i - t_{i-1}$, $t_0 = 0$. The number of individuals alive at the beginning of the interval may be estimated by $P_i = M_i + (1 - a_i)d_i$ where $a_i$ is the probability that an individual dying in the interval dies prior to the interval midpoint. For cohort table, $P_i$ is input directly while the death rate in the interval, $r_i$, is not needed.

The probability that an individual dies during the age interval from $t_{i-1}$ to $t_i$ is given by $q_i = d_i/P_i$. It is assumed that all individuals alive at the beginning of the last interval die during the last interval. Thus, $q_n = 1.0$. The asymptotic variance of $q_i$ can be estimated by

$$\sigma^2_i = q_i(1 - q_i)/P_i$$

For a population table, the number of individuals alive in the middle of the time interval (input in nCohort [i]) must be adjusted to correspond to the number of deaths observed in the interval. The algorithm assumes that the number of deaths observed in interval $h_i$ occur over a time period equal to $h_i$. If $d_i$ is measured over a period $u_i$, where $u_i \neq d_i$, then nCohort [i] must be adjusted to correspond to $d_i$ by multiplication by $u_i/h_i$, i.e., the value $M_i$ input as nCohort [i] is computed as

$$M_i^* = M_i u_i/h_i$$

Let $S_i$ denote the number of survivors at time $t_i$ from a hypothetical (for population table) or observed (for cohort table) population. Then, $S_0 =$ initialPopulation for population table, and $S_0 =$ nCohort [0] for cohort table, and $S_i$ is given by $S_i = S_{i-1} - \delta_{i-1}$ where $\delta_i = S_i q_i$ is the number of individuals who die in the $i$th interval. The proportion of survivors in the interval is given by $V_i = S_i/S_0$ while the asymptotic variance of $V_i$ can be estimated as follows:

$$\text{var}(V_i) = V_i^2 \sum_{j=1}^{i-1} \frac{\sigma_j^2}{(1 - q_j)^2}$$

The expected lifetime at the beginning of the interval is calculated as the total lifetime remaining for all survivors alive at the beginning of the interval divided by the number of survivors at the beginning of the interval. If $e_i$ denotes this average expected lifetime, then the variance of $e_i$ can be estimated as (see Chiang 1968):

$$\text{var}(e_i) = \frac{\sum_{j=i}^{n-1} p_j^2 \sigma_j^2 [e_{j+1} + h_{j+1}(1 - a_j)]^2}{p_j^2}$$

where $\text{var}(e_0) = 0.0$. 

---

**Survival and Reliability Analysis**  
**LifeTables • 1093**
Finally, the total number of time units lived by all survivors in the time interval can be estimated as:

\[ U_i = h_i[S_i - \delta_i(1 - a_i)] \]

**Property**

**PopulationSize**

```csharp
virtual public int PopulationSize {get; set; }
```

**Description**

Sets the population size at the beginning of the first age interval in requesting a population table.

**Property Value**

An int scalar specifying the initial population.

Default: A value of 10,000 is used to allow easy entry of `nCohorts` and `nDeaths` when numbers are available as percentages.

**Constructor**

```csharp
public LifeTables(int[] nCohort, double[] age, double[] a)
```

**Description**

Constructs a new `LifeTables` instance. The number of classes, `nClasses` is equal to the length of the input array `nCohort`.

**Parameters**

- `nCohort` – An int array of length `nClasses` containing the cohort sizes during each interval. If the Population Table option is used, then `nCohort[i]` contains the size of the population at the midpoint of interval `i`. Otherwise, `nCohort[i]` contains the size of the cohort at the beginning of interval `i`. When requesting a population table, the population sizes in `nCohort` may need to be adjusted to correspond to the number of deaths in `nDeaths`. See the class description section for more information.

- `age` – A double array of length `nClasses + 1` containing the lowest age in each age interval, and in `age[nClasses]`, the endpoint of the last age interval. Negative `age[0]` indicates that the age intervals are all of length `|age[0]|` and that the initial age interval is from 0.0 to `|age[0]|`. In this case, all other elements of `age` need not be specified. `age[nClasses]` need not be specified when getting a cohort table.

- `a` – A double array of length `nClasses` containing the fraction of those dying within each interval who die before the interval midpoint. A common choice for all `a[i]` is 0.5. This choice may also be specified by setting `a[0]` to any negative value. In this case, the remaining values of `a` need not be specified.
Methods

GetLifeTable
virtual public double[,] GetLifeTable()

Description
Compute a cohort table.

Returns
A double matrix of dimensions nClasses by 12 containing the population table. Entries in the $i$th row are for the age interval defined by age[i]. Column definitions are described in the following table.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Lowest age in the age interval.</td>
</tr>
<tr>
<td>1</td>
<td>Fraction of those dying within the interval who die before the interval midpoint.</td>
</tr>
<tr>
<td>2</td>
<td>Number surviving to the beginning of the interval.</td>
</tr>
<tr>
<td>3</td>
<td>Number of deaths in the interval.</td>
</tr>
<tr>
<td>4</td>
<td>Death rate in the interval. For cohort table, this column is set to NaN (not a number).</td>
</tr>
<tr>
<td>5</td>
<td>Proportion dying in the interval.</td>
</tr>
<tr>
<td>6</td>
<td>Standard error of the proportion dying in the interval.</td>
</tr>
<tr>
<td>7</td>
<td>Proportion of survivors at the beginning of the interval.</td>
</tr>
<tr>
<td>8</td>
<td>Standard error of the proportion of survivors at the beginning of the interval.</td>
</tr>
<tr>
<td>9</td>
<td>Expected lifetime at the beginning of the interval.</td>
</tr>
<tr>
<td>10</td>
<td>Standard error of the expected life at the beginning of the interval.</td>
</tr>
<tr>
<td>11</td>
<td>Total number of time units lived by all of the population in the interval.</td>
</tr>
</tbody>
</table>

GetPopulationTable
virtual public double[,] GetPopulationTable(int[] nDeaths)

Description
Compute a population table.

Parameter
nDeaths – An int array of nClasses containing the number of deaths in each age interval.

Returns
A double matrix of dimensions nClasses by 12 containing the population table. Entries in the $i$th row are for the age interval defined by age[i]. Column definitions are the same as in GetLifeTable.
Example: Life Tables

This example is taken from Chiang (1968). The cohort life table has thirteen equally spaced intervals, so 
age[0] is set to -5.0. Similarly, the probabilities of death prior to the middle of the interval are all taken to 
be 0.5, so a[0] is set to -1.0.

```csharp
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;

public class LifeTablesEx1
{
    public static void Main(String[] args)
    {
        int[] nCohort = {270, 268, 264, 261, 254, 251, 248, 232,
                         166, 130, 76, 34, 13};
        double[] age = new double[nCohort.Length + 1];
        double[] a = new double[nCohort.Length];

        age[0] = -5.0;
        a[0] = -1.0;

        LifeTables lt = new LifeTables(nCohort, age, a);
        double[,] table = lt.GetLifeTable();

        PrintMatrixFormat pmf = new PrintMatrixFormat();
        String[] cols = {"Age", "PDHALF", "Alive", 
                         "Deaths", "Death Rate", "P(D)", "Std(P(D))", "P(S)", "Std(P(S))", 
                         "Lifetime", "Std(Life)", "Time Units"};
        pmf.SetColumnLabels(cols);
        pmf.NumberFormat = "0.0####";
        PrintMatrix pm = new PrintMatrix("Life Table");
        pm.SetPageWidth(40);
        pm.Print(pmf, table);
    }
}
```

Output

```
<table>
<thead>
<tr>
<th>Age</th>
<th>PDHALF</th>
<th>Alive</th>
<th>Deaths</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
<td>270.0</td>
<td>2.0</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>268.0</td>
<td>4.0</td>
</tr>
<tr>
<td>2</td>
<td>10.0</td>
<td>264.0</td>
<td>3.0</td>
</tr>
<tr>
<td>3</td>
<td>15.0</td>
<td>261.0</td>
<td>7.0</td>
</tr>
<tr>
<td>4</td>
<td>20.0</td>
<td>254.0</td>
<td>3.0</td>
</tr>
<tr>
<td>5</td>
<td>25.0</td>
<td>251.0</td>
<td>3.0</td>
</tr>
<tr>
<td>6</td>
<td>30.0</td>
<td>248.0</td>
<td>16.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
<td>232.0</td>
<td>66.0</td>
</tr>
<tr>
<td>8</td>
<td>40.0</td>
<td>166.0</td>
<td>36.0</td>
</tr>
<tr>
<td>9</td>
<td>45.0</td>
<td>130.0</td>
<td>54.0</td>
</tr>
<tr>
<td>10</td>
<td>50.0</td>
<td>76.0</td>
<td>42.0</td>
</tr>
<tr>
<td>11</td>
<td>55.0</td>
<td>34.0</td>
<td>21.0</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>Death Rate</th>
<th>P(D)</th>
<th>Std(P(D))</th>
<th>P(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NaN</td>
<td>0.00741</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>NaN</td>
<td>0.01493</td>
<td>0.99259</td>
</tr>
<tr>
<td>2</td>
<td>NaN</td>
<td>0.01136</td>
<td>0.97778</td>
</tr>
<tr>
<td>3</td>
<td>NaN</td>
<td>0.02682</td>
<td>0.96667</td>
</tr>
<tr>
<td>4</td>
<td>NaN</td>
<td>0.01181</td>
<td>0.94074</td>
</tr>
<tr>
<td>5</td>
<td>NaN</td>
<td>0.01195</td>
<td>0.92963</td>
</tr>
<tr>
<td>6</td>
<td>NaN</td>
<td>0.06452</td>
<td>0.91852</td>
</tr>
<tr>
<td>7</td>
<td>NaN</td>
<td>0.28448</td>
<td>0.85926</td>
</tr>
<tr>
<td>8</td>
<td>NaN</td>
<td>0.21687</td>
<td>0.61481</td>
</tr>
<tr>
<td>9</td>
<td>NaN</td>
<td>0.41538</td>
<td>0.48148</td>
</tr>
<tr>
<td>10</td>
<td>NaN</td>
<td>0.55263</td>
<td>0.28148</td>
</tr>
<tr>
<td>11</td>
<td>NaN</td>
<td>0.61765</td>
<td>0.12593</td>
</tr>
<tr>
<td>12</td>
<td>NaN</td>
<td>1.0</td>
<td>0.04815</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Std(P(S))</th>
<th>Lifetime</th>
<th>Std(Life)</th>
<th>Time Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>43.18519</td>
<td>1345.0</td>
</tr>
<tr>
<td>1</td>
<td>0.00522</td>
<td>38.48881</td>
<td>1330.0</td>
</tr>
<tr>
<td>2</td>
<td>0.00897</td>
<td>34.03409</td>
<td>1312.5</td>
</tr>
<tr>
<td>3</td>
<td>0.01092</td>
<td>29.39655</td>
<td>1287.5</td>
</tr>
<tr>
<td>4</td>
<td>0.01437</td>
<td>25.1378</td>
<td>1262.5</td>
</tr>
<tr>
<td>5</td>
<td>0.01557</td>
<td>20.40387</td>
<td>1247.5</td>
</tr>
<tr>
<td>6</td>
<td>0.01668</td>
<td>15.625</td>
<td>1200.0</td>
</tr>
<tr>
<td>7</td>
<td>0.02116</td>
<td>11.53017</td>
<td>995.0</td>
</tr>
<tr>
<td>8</td>
<td>0.02962</td>
<td>10.12048</td>
<td>740.0</td>
</tr>
<tr>
<td>9</td>
<td>0.03041</td>
<td>7.23077</td>
<td>515.0</td>
</tr>
<tr>
<td>10</td>
<td>0.02737</td>
<td>5.59211</td>
<td>275.0</td>
</tr>
<tr>
<td>11</td>
<td>0.02019</td>
<td>4.41176</td>
<td>117.5</td>
</tr>
<tr>
<td>12</td>
<td>0.01303</td>
<td>2.5</td>
<td>32.5</td>
</tr>
</tbody>
</table>
Chapter 21: Probability
Distribution Functions
and Inverses

Types

class Cdf ................................................................. 1101
class InvCdf ............................................................ 1137
class Pdf ................................................................. 1149
interface ICdfFunction ............................................... 1165
class InverseCdf ......................................................... 1165
class IDistribution ..................................................... 1167
class IProbabilityDistribution ..................................... 1168
class GammaDistribution ........................................... 1170
class NormalDistribution ........................................... 1170
class LogNormalDistribution ...................................... 1175
class PoissonDistribution .......................................... 1177

Usage Notes

Definitions and discussions of the terms basic to this chapter can be found in Johnson and Kotz (1969, 1970a, 1970b). These are also good references for the specific distributions.

In order to keep the calling sequences simple, whenever possible, the methods/classes described in this chapter are written for standard forms of statistical distributions. Hence, the number of parameters for any given distribution may be fewer than the number often associated with the distribution. Also, the methods relating to the normal distribution, Cdf.Normal, InvCdf.Normal, and Pdf.Normal are for a normal distribution with mean equal to zero and variance equal to one. For other means and variances, it is very easy for the user to standardize the variables by subtracting the mean and dividing by the square root of the variance.

The distribution function for the (real, single-valued) random variable X is the function F defined for all
real $x$ by

$$F(x) = \text{Prob}(X \leq x)$$

where $\text{Prob}(\cdot)$ denotes the probability of an event. The distribution function is often called the 
\textit{cumulative distribution function} (CDF).

For distributions with finite ranges, such as the beta distribution, the CDF is 0 for values less than the left
endpoint and 1 for values greater than the right endpoint. The methods in the Cdf, InvCdf, and Pdf
classes described in this chapter return the correct values for the distribution functions when values
outside of the range of the random variable are input, but warning error conditions are set in these cases.

\textbf{Discrete Random Variables}

For discrete distributions, the function giving the probability that the random variable takes on specific
values is called the \textit{probability function}, defined by

$$p(x) = \text{Prob}(X = x)$$

The CDF for a discrete random variable is

$$F(x) = \sum_A p(k)$$

where $A$ is set such that $k \leq x$. Since the distribution function is a step function, its inverse does not exist
uniquely.

\textbf{Continuous Distributions}

For continuous distributions, a probability function, as defined above, would not be useful because the
probability of any given point is 0. For such distributions, the useful analog is the \textit{probability density
function} (PDF). The integral of the PDF is the probability over the interval, if the continuous random
variable $X$ has PDF $f$, then

$$\text{Prob}(a \leq X \leq b) = \int_a^b f(x) \, dx$$

The relationship between the CDF and the PDF is

$$F(x) = \int_{-\infty}^x f(t) \, dt$$

For (absolutely) continuous distributions, the value of $F(x)$ uniquely determines $x$ within the support of
the distribution. The methods in the InvCdf class compute the inverses of the distribution functions.
That is, given $F(x)$, a method such as Beta in the InvCdf class computes $x$. The inverses are defined
only over the open interval (0,1).
Additional Comments

Whenever a probability close to 1.0 results from a call to a distribution function or is to be input to an inverse function, it is often impossible to achieve good accuracy because of the nature of the representation of numeric values. In this case, it may be better to work with the complementary distribution function (one minus the distribution function). If the distribution is symmetric about some point (as the normal distribution, for example) or is reflective about some point (as the beta distribution, for example), the complementary distribution function has a simple relationship with the distribution function. For example, to evaluate the standard normal distribution at 4.0, using the Normal method in the Cdf class directly, the result to six places is 0.999968. Only two of those digits are really useful, however. A more useful result may be 1.000000 minus this value, which can be obtained to six places as 3.16712e-05 by evaluating Normal at -4.0. For the normal distribution, the two values are related by \( \Phi(x) = 1 - \Phi(-x) \), where \( \Phi(\cdot) \) is the normal distribution function. Another example is the beta distribution with parameters 2 and 10. This distribution is skewed to the right, so evaluating Cdf.Beta at 0.7, 0.999953 is obtained. A more precise result is obtained by evaluating Cdf.Beta with parameters 10 and 2 at 0.3. This yields 4.72392e-5.

Many of the algorithms used by the classes in this chapter are discussed by Abramowitz and Stegun (1964). The algorithms make use of various expansions and recursive relationships and often use different methods in different regions.

Cumulative distribution functions are defined for all real arguments. However, if the input to one of the distribution functions in this chapter is outside the range of the random variable, an error is issued.

---

Cdf Class

```java
public class Imsl.Stat.Cdf
Cumulative probability distribution functions.
```

See Also

Probability density function (p. 1149), Inverse Cumulative Distribution Function (p. 1137)

Methods

**Beta**

```java
static public double Beta(double x, double pin, double qin)
```

---

Probability Distribution Functions and Inverses   Cdf • 1101
Description
Evaluates the beta cumulative probability distribution function.

Parameters
- x – A double, the argument at which the function is to be evaluated.
- pin – A double, the first beta distribution parameter.
- qin – A double, the second beta distribution parameter.

Returns
A double, the probability that a beta random variable takes on a value less than or equal to x.

Remarks
Method Cdf.Beta evaluates the distribution function of a beta random variable with parameters pin and qin. This function is sometimes called the incomplete beta ratio and, with p = pin and q = qin, is denoted by \( I_x(p,q) \). It is given by

\[
I_x(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)} \int_0^x t^{p-1}(1-t)^{q-1} \, dt
\]

where \( \Gamma(\cdot) \) is the gamma function. The value of the distribution function \( I_x(p,q) \) is the probability that the random variable takes a value less than or equal to x.

The integral in the expression above is called the incomplete beta function and is denoted by \( \beta_x(p,q) \).

The constant in the expression is the reciprocal of the beta function (the incomplete function evaluated at one) and is denoted by \( \beta_1(p,q) \).

**BetaMean**

static public double BetaMean(double pin, double qin)

**Description**
Evaluates the mean of the beta cumulative probability distribution function.

**Parameters**
- pin – A double, the first beta distribution parameter.
- qin – A double, the second beta distribution parameter.

**Returns**
A double, the mean of the beta distribution function.

**BetaVariance**

static public double BetaVariance(double pin, double qin)
Description
Evaluates the variance of the beta cumulative probability distribution function.

Parameters
- pin – A double, the first beta distribution parameter.
- qin – A double, the second beta distribution parameter.

Returns
A double, the variance of the beta distribution function.

Binomial
static public double Binomial(int k, int n, double pin)

Description
Evaluates the binomial cumulative probability distribution function.

Parameters
- k – The int argument for which the binomial distribution function is to be evaluated.
- n – The int number of Bernoulli trials.
- pin – A double scalar value representing the probability of success on each independent trial.

Returns
A double scalar value representing the probability that a binomial random variable takes a value less than or equal to k. This value is the probability that k or fewer successes occur in n independent Bernoulli trials, each of which has a pin probability of success.

Remarks
Method Cdf.Binomial evaluates the distribution function of a binomial random variable with parameters n and p with p=pin. It does this by summing probabilities of the random variable taking on the specific values in its range. These probabilities are computed by the recursive relationship

\[ \Pr(X = j) = \frac{(n+1-j)p}{j(1-p)} \Pr(X = j-1) \]

To avoid the possibility of underflow, the probabilities are computed forward from 0, if k is not greater than n times p, and are computed backward from n, otherwise. The smallest positive machine number, \( \varepsilon \), is used as the starting value for summing the probabilities, which are rescaled by \((1-p)\varepsilon\) if forward computation is performed and by \(p\varepsilon\) if backward computation is done. For the special case of \( p = 0 \), Cdf.Binomial is set to 1; and for the case \( p = 1 \), Cdf.Binomial is set to 1 if \( k = n \) and to 0 otherwise.
Parameters

- $x$ – The $x$-coordinate of the point for which the bivariate normal distribution function is to be evaluated.
- $y$ – The $y$-coordinate of the point for which the bivariate normal distribution function is to be evaluated.
- $\rho$ – The correlation coefficient.

Returns

The probability that a bivariate normal random variable $(X, Y)$ with correlation $\rho$ satisfies $X \leq x$ and $Y \leq y$.

Remarks

Let $(X, Y)$ be a bivariate normal variable with mean $(0, 0)$ and variance-covariance matrix

$$
\begin{bmatrix}
1 & \rho \\
\rho & 1
\end{bmatrix}
$$

This method computes the probability that $X \leq x$ and $Y \leq y$. 


Chi

```csharp
static public double Chi(double chsq, double df)
```

**Description**

Evaluates the chi-squared cumulative probability distribution function.

**Parameters**

- `chsq` – A `double` specifying the argument at which the function is to be evaluated.
- `df` – A `double` specifying the number of degrees of freedom. This must be at least 0.5.

**Returns**

A `double` specifying the probability that a chi-squared random variable takes a values less than or equal to `chsq`.

---

**Bivariate Normal Distribution Function**

![Bivariate Normal Distribution Function](image)

-5.00 -3.00 -1.00 1.00 3.00 5.00

0.00 0.20 0.40 0.60 0.80 1.00

bivariateNormal(x,y,.9)

0.0 1.0 2.0 3.0
Remarks
Method Cdf.Chi evaluates the distribution function, \( F \), of a chi-squared random variable with \( df \) degrees of freedom, that is, with \( v = df \), and \( x = \text{chsq} \).

\[
F(x) = \frac{1}{2^{\nu/2} \Gamma(\nu/2)} \int_0^x e^{-t/2} t^{\nu/2-1} dt
\]

where \( \Gamma(\cdot) \) is the gamma function. The value of the distribution function at the point \( x \) is the probability that the random variable takes a value less than or equal to \( x \).

For \( v > 343 \), Cdf.Chi uses the Wilson-Hilferty approximation (Abramowitz and Stegun 1964, equation 26.4.17) to the normal distribution, and method Cdf.Normal is used to evaluate the normal distribution function.

For \( v \leq 343 \), Cdf.Chi uses series expansions to evaluate the distribution function. If \( x < \max(v/2, 26) \), Cdf.Chi uses the series 6.5.29 in Abramowitz and Stegun (1964), otherwise, it uses the asymptotic expansion 6.5.32 in Abramowitz and Stegun.
For greater right tail accuracy, see Cdf.ComplementaryChi (p. 1108).

**ChiMean**
static public double ChiMean(double df)

**Description**
Evaluates the mean of the chi-squared cumulative probability distribution function.

**Parameter**
- df – A double scalar value representing the number of degrees of freedom. This must be at least 0.5.

**Returns**
A double, the mean of the chi-squared distribution function.

**ChiVariance**
static public double ChiVariance(double df)

**Description**
Evaluates the variance of the chi-squared cumulative probability distribution function.

**Parameter**
- df – A double scalar value representing the number of degrees of freedom. This must be at least 0.5.

**Returns**
A double, the variance of the chi-squared distribution function.

**ComplementaryChi**
static public double ComplementaryChi(double chsq, double df)

**Description**
Calculates the complement of the chi-squared distribution.

**Parameters**
- chsq – A double scalar value at which $Pr(x > \chi^2)$ is to be evaluated.
- df – A double specifying the number of degrees of freedom. This must be at least 0.5.

**Returns**
A double specifying the probability that a chi-squared random variable takes a value greater than chsq.

**Remarks**
Method Cdf.ComplementaryChi evaluates the distribution function, $1 - F$, of a chi-squared random variable with df degrees of freedom, that is, with $v = df$, and $x = chsq$.

$$F(x) = \frac{1}{2^{v/2}\Gamma(v/2)} \int_0^x e^{-t/2}t^{v/2-1}dt$$
where $\Gamma(\cdot)$ is the gamma function. The value of the distribution function at the point $x$ is the probability that the random variable takes a value greater than $x$.

For $v > 343$, `Cdf.ComplementaryChi` uses the Wilson-Hilferty approximation (Abramowitz and Stegun 1964, equation 26.4.17) to the one minus the normal distribution, and method `Cdf.Normal` is used to evaluate the normal distribution function.

For $v \leq 343$, `Cdf.ComplementaryChi` uses series expansions to evaluate the distribution function. If $x < \max(v/2, 26)$, `Cdf.ComplementaryChi` uses the series 6.5.29 in Abramowitz and Stegun (1964), otherwise, it uses the asymptotic expansion 6.5.32 in Abramowitz and Stegun. This function provides higher right tail accuracy for the Chi-squared distribution.

**ComplementaryF**

static public double ComplementaryF(double x, double dfn, double dfd)
**Description**
Calculates the complement of the F distribution function.

**Parameters**
- \(x\) – A double, the argument at which \(Pr(x > F)\) is to be evaluated.
- \(dfn\) – A double, the numerator degrees of freedom. It must be positive.
- \(dfd\) – A double, the denominator degrees of freedom. It must be positive.

**Returns**
A double, the probability that an \(F\) random variable takes on a value greater than \(x\).

**Remarks**
\texttt{Cdf.ComplementaryF} evaluates one minus the distribution function of a Snedecor’s \(F\) random variable with \(dfn\) numerator degrees of freedom and \(dfd\) denominator degrees of freedom. The function is evaluated by making a transformation to a beta random variable and then using the function \texttt{Cdf.Beta}. If \(X\) is an \(F\) variate with \(v_1\) and \(v_2\) degrees of freedom and \(Y = v_1 X / (v_2 + v_1 X)\), then \(Y\) is a beta variate with parameters \(p = v_1 / 2\) and \(q = v_2 / 2\). \texttt{Cdf.ComplementaryF} also uses a relationship between \(F\) random variables that can be expressed as follows:

\[
F(X, dfn, dfd) = F(1/X, dfd, dfn)
\]

This function provides higher right tail accuracy for the \(F\) distribution.
ComplementaryStudentsT

static public double ComplementaryStudentsT(double t, double df)

Description
Calculates the complement of the Student's $t$ distribution.

Parameters
- $t$ – A double scalar value for which $Pr(x > t)$ is to be evaluated.
- $df$ – A double scalar value representing the number of degrees of freedom. This must be at least one.

Returns
A double scalar value representing the probability that a Student's $t$ random variable takes a value
greater than $t$.

### Remarks

Method `Cdf.ComplementaryStudentsT` evaluates one minus the distribution function of a Student’s $t$ random variable with $df$ degrees of freedom. If the square of $t$ is greater than or equal to $df$, the relationship of a $t$ to an $f$ random variable (and subsequently, to a beta random variable) is exploited, and routine `Cdf.Beta` is used. Otherwise, the method described by Hill (1970) is used. If $df$ is not an integer, if $df$ is greater than 19, or if $df$ is greater than 200, a Cornish-Fisher expansion is used to evaluate the distribution function. If $df$ is less than 20 and $|t|$ is less than 2.0, a trigonometric series (see Abramowitz and Stegun 1964, equations 26.7.3 and 26.7.4, with some rearrangement) is used. For the remaining cases, a series given by Hill (1970) that converges well for large values of $t$ is used. This function provides higher right tail accuracy for the Student’s $t$ distribution.
DiscreteUniform
dynamic public double DiscreteUniform(int x, int n)

Description
Evaluates the discrete uniform cumulative probability distribution function.

Parameters
- x – An int scalar value representing the argument at which the function is to be evaluated.
- n – An int scalar value representing the upper limit of the discrete uniform distribution.

Returns
A double scalar value representing the probability that a discrete uniform random variable takes a value less than or equal to x.

Exponential
dynamic public double Exponential(double x, double scale)

Description
Evaluates the exponential cumulative probability distribution function.

Parameters
- x – A double scalar value representing the argument at which the function is to be evaluated.
- scale – A double scalar value representing the scale parameter, b.

Returns
A double scalar value representing the probability that an exponential random variable takes on a value less than or equal to x.

Remarks
Method Cdf.Exponential is a special case of the gamma distribution function, which evaluates the distribution function, F, with scale parameter b and shape parameter a, used in the gamma distribution function equal to 1.0. That is,

\[ F(x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t/b} dt \]

where \( \Gamma(\cdot) \) is the gamma function. (The gamma function is the integral from 0 to \( \infty \) of the same integrand as above). The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x.

If x is less than or equal to 1.0, Cdf.Gamma uses a series expansion. Otherwise, a continued fraction expansion is used. (See Abramowitz and Stegun, 1964.)
ExtremeValue
static public double ExtremeValue(double x, double mu, double beta)

Description
Evaluates the extreme value cumulative probability distribution function.

Parameters
  x – A double scalar value representing the argument at which the function is to be evaluated.
  mu – A double scalar value representing the location parameter, \( \mu \).
  beta – A double scalar value representing the scale parameter, \( \beta \).

Returns
A double scalar value representing the probability that an extreme value random variable takes on a value less than or equal to \( x \).
Remarks
Method Cdf.ExtremeValue, also known as the Gumbel minimum distribution, evaluates the extreme value distribution function, \( F \), of a uniform random variable with location parameter \( \mu \) and shape parameter \( \beta \); that is,

\[
F(x) = \int_0^x 1 - e^{-\left(\frac{x-\mu}{\beta}\right)} \, dt
\]

The case where \( \mu = 0 \) and \( \beta = 1 \) is called the standard Gumbel distribution.

Random numbers are generated by evaluating uniform variates \( u_i \), equating the continuous distribution function, and then solving for \( x_i \) by first computing \( \frac{x_i-\mu}{\beta} = \log(-\log(1-u_i)) \).

\[
F \text{ static public double } F(\text{double } x, \text{ double } dfn, \text{ double } dfd)
\]
Description
Evaluates the F cumulative probability distribution function.

Parameters
- \( x \) – A double, the argument at which the function is to be evaluated.
- \( dfn \) – A double, the numerator degrees of freedom. It must be positive.
- \( dfd \) – A double, the denominator degrees of freedom. It must be positive.

Returns
A double, the probability that an F random variable takes on a value less than or equal to \( x \).

Remarks
Cdf.F evaluates the distribution function of a Snedecor’s F random variable with \( dfn \) numerator degrees of freedom and \( dfd \) denominator degrees of freedom. The function is evaluated by making a transformation to a beta random variable and then using the function Cdf.Beta. If \( X \) is an F variate with \( v_1 \) and \( v_2 \) degrees of freedom and \( Y = v_1 X / (v_2 + v_1 X) \), then \( Y \) is a beta variate with parameters \( p = v_1 / 2 \) and \( q = v_2 / 2 \). F also uses a relationship between F random variables that can be expressed as follows:

\[
F(X, dfn, dfd) = F(1/X, dfd, dfn)
\]

For greater right tail accuracy, see Cdf.ComplementaryF (p. 1109).
Gamma

static public double Gamma(double x, double a)

Description
Evaluates the gamma cumulative probability distribution function.

Parameters
- x – A double scalar value representing the argument at which the function is to be evaluated.
- a – A double scalar value representing the shape parameter. This must be positive.

Returns
A double scalar value representing the probability that a gamma random variable takes on a value less than or equal to x.
Remarks

Method Cdf.Gamma evaluates the distribution function, \( F \), of a gamma random variable with shape parameter \( a \); that is,

\[
F(x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t/t^{a-1}} dt
\]

where \( \Gamma(\cdot) \) is the gamma function. (The gamma function is the integral from 0 to \( \infty \) of the same integrand as above). The value of the distribution function at the point \( x \) is the probability that the random variable takes a value less than or equal to \( x \).

The gamma distribution is often defined as a two-parameter distribution with a scale parameter \( b \) (which must be positive), or even as a three-parameter distribution in which the third parameter \( c \) is a location parameter. In the most general case, the probability density function over \((c, \infty)\) is

\[
f(t) = \frac{1}{b^a \Gamma(a)} e^{-[(t-c)/b]} (t-c)^{a-1}
\]

If \( T \) is such a random variable with parameters \( a, b \), and \( c \), the probability that \( T \leq t_0 \) can be obtained from Cdf.Gamma by setting \( X = (t_0 - c)/b \).

If \( X \) is less than \( a \) or if \( X \) is less than or equal to 1.0, Cdf.Gamma uses a series expansion. Otherwise, a continued fraction expansion is used. (See Abramowitz and Stegun, 1964.)
Geometric
static public double Geometric(int x, double pin)

Description
Evaluates the discrete geometric cumulative probability distribution function.

Parameters
  x – An int scalar value representing the argument at which the function is to be evaluated.
  pin – A double scalar value representing the probability parameter for each independent trial (the probability of success for each independent trial).

Returns
A double scalar value representing the probability that a geometric random variable takes a value less than or equal to x. The return value is the probability that up to x trials would be observed before observing a success.
**Hypergeometric**

*static public double Hypergeometric(int k, int sampleSize, int defectivesInLot, int lotSize)*

**Description**

Evaluates the hypergeometric cumulative probability distribution function.

**Parameters**

- `k` – An int, the argument at which the function is to be evaluated.
- `sampleSize` – An int, the sample size, `n`.
- `defectivesInLot` – An int, the number of defectives in the lot, `m`.
- `lotSize` – An int, the lot size, `l`.

**Returns**

A double, the probability that a hypergeometric random variable takes a value less than or equal to `k`.

**Remarks**

Method *Cdf.Hypergeometric* evaluates the distribution function of a hypergeometric random variable with parameters `n`, `l`, and `m`. The hypergeometric random variable `X` can be thought of as the number of items of a given type in a random sample of size `n` that is drawn without replacement from a population of size `l` containing `m` items of this type. The probability function is

\[
\Pr(X = j) = \binom{m}{j} \binom{n-j}{i-j} \binom{l}{n} \quad \text{for } j = i, i+1, i+2, \ldots, \min(n, m)
\]

where \( i = \max(0, n - l + m) \).

If `k` is greater than or equal to `i` and less than or equal to \( \min(n, m) \), `Cdf.Hypergeometric` sums the terms in this expression for `j` going from `i` up to `k`. Otherwise, `Cdf.Hypergeometric` returns 0 or 1, as appropriate. So, as to avoid rounding in the accumulation, `Cdf.Hypergeometric` performs the summation differently depending on whether or not `k` is greater than the mode of the distribution, which is the greatest integer less than or equal to \( (m + 1)(n + 1)/(l + 2) \).

**Logistic**

*static public double Logistic(double x, double mu, double s)*

**Description**

Evaluates the logistic cumulative probability distribution function.

**Parameters**

- `x` – A double scalar value representing the argument at which the function is to be evaluated.
- `mu` – A double scalar value representing the location parameter, \( \mu \).
- `s` – A double scalar value representing the scale parameter.

**Returns**

A double scalar value representing the probability that a logistic random variable takes a value less than or equal to `x`. 

IMSL C# Numerical Library
Remarks
Method Cdf.Logistic evaluates the distribution function, $F$, of a logistic random variable with location parameter $\mu$ and scale parameter $s$. It is given by

$$F(x, \mu, s) = \frac{1}{1 + e^{-(x-\mu)/s}}$$

where $s > 0$.

LogNormal
static public double LogNormal(double x, double mu, double sigma)

Description
Evaluates the standard lognormal cumulative probability distribution function.

Parameters
- $x$ – A double scalar value representing the argument at which the function is to be evaluated.
- $\mu$ – A double scalar value representing the location parameter.
- $\sigma$ – A double scalar value representing the shape parameter. $\sigma$ must be a positive.

Returns
A double scalar value representing the probability that a standard lognormal random variable takes a value less than or equal to $x$.

Remarks

$$F(x) = \frac{1}{x\sigma\sqrt{2\pi}} \int_0^{\ln(x/\mu)} e^{-\frac{t^2}{2\sigma^2}} dt$$
NoncentralBeta

static public double NoncentralBeta(double x, double shape1, double shape2, double lambda)

Description
Evaluates the noncentral beta cumulative probability distribution function (CDF).

Parameters

x – A double scalar value representing the argument at which the function is to be evaluated. x must be nonnegative and less than or equal to 1.

shape1 – A double scalar value representing the first shape parameter. shape1 must be positive.

shape2 – A double scalar value representing the second shape parameter. shape2 must be positive.
\( \lambda \) – A double scalar value representing the noncentrality parameter. \( \lambda \) must be nonnegative.

**Returns**

A double scalar value representing the probability that a noncentral beta random variable takes a value less than or equal to \( x \).

**Remarks**

The noncentral beta distribution is a generalization of the beta distribution. If \( Z \) is a noncentral chi-square random variable with noncentrality parameter \( \lambda \) and \( 2\alpha_1 \) degrees of freedom, and \( Y \) is a chi-square random variable with \( 2\alpha_2 \) degrees of freedom which is statistically independent of \( Z \), then

\[
X = \frac{Z}{Z + Y} = \frac{\alpha_1 F}{\alpha_1 F + \alpha_2}
\]

is a noncentral beta-distributed random variable and

\[
F = \frac{\alpha_2 Z}{\alpha_1 Y} = \frac{\alpha_2 X}{\alpha_1 (1 - X)}
\]

is a noncentral \( F \)-distributed random variable. The CDF for noncentral beta variable \( X \) can thus be simply defined in terms of the noncentral \( F \) CDF:

\[
CDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda) = CDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda)
\]

where \( CDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda) \) is the noncentral beta CDF with \( x = x, \alpha_1 = \text{shape1}, \alpha_2 = \text{shape2}, \) and noncentrality parameter \( \lambda = \lambda \); \( CDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda) \) is the noncentral \( F \) CDF with argument \( f \), numerator and denominator degrees of freedom \( 2\alpha_1 \) and \( 2\alpha_2 \) respectively, and noncentrality parameter \( \lambda \); and:

\[
f = \frac{\alpha_2 x}{\alpha_1 (1 - x)}, \quad x = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2}
\]

(See documentation for class Cdf method NoncentralF for a discussion of how the noncentral \( F \) CDF is defined and calculated.)

With a noncentrality parameter of zero, the noncentral beta distribution is the same as the beta distribution.

**Noncentralchi**

static public double Noncentralchi(double chsq, double df, double alam)

**Description**

Evaluates the noncentral chi-squared cumulative probability distribution function.

**Parameters**

- **chsq** – A double scalar value representing the argument at which the function is to be evaluated.
- **df** – A double scalar value representing the number of degrees of freedom. \( df \) must be positive.
- **alam** – A double scalar value representing the noncentrality parameter. This must be nonnegative, and \( \text{alam} + df \) must be less than or equal to 200,000.
Returns
A double scalar value representing the probability that a chi-squared random variable takes a value less than or equal to \(chsq\).

Remarks
Method Cdf.Noncentralchi evaluates the distribution function, \(F\), of a noncentral chi-squared random variable with \(df\) degrees of freedom and noncentrality parameter \(alam\), that is, with \(v = df, \lambda = alam\), and \(\chi = chsq\).

\[
F(x) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} \int_{0}^{x} \frac{t^{(v+2i)/2-1}e^{-t/2}}{2^{(v+2i)/2} \Gamma \left( \frac{v+2i}{2} \right)} dt
\]

where \(\Gamma(\cdot)\) is the gamma function. This is a series of central chi-squared distribution functions with Poisson weights. The value of the distribution function at the point \(x\) is the probability that the random variable takes a value less than or equal to \(x\).

The noncentral chi-squared random variable can be defined by the distribution function above, or alternatively and equivalently, as the sum of squares of independent normal random variables. If the \(Y_i\) have independent normal distributions with means \(\mu_i\) and variances equal to one and

\[
\chi = \sum_{i=1}^{n} Y_i^2
\]

then \(\chi\) has a noncentral chi-squared distribution with \(n\) degrees of freedom and noncentrality parameter equal to

\[
\sum_{i=1}^{n} \mu_i^2
\]

With a noncentrality parameter of zero, the noncentral chi-squared distribution is the same as the chi-squared distribution.

Cdf.Noncentralchi determines the point at which the Poisson weight is greatest, and then sums forward and backward from that point, terminating when the additional terms are sufficiently small or when a maximum of 1000 terms have been accumulated. The recurrence relation 26.4.8 of Abramowitz and Stegun (1964) is used to speed the evaluation of the central chi-squared distribution functions.
NoncentralF
static public double NoncentralF(double f, double df1, double df2, double lambda)

Description
Evaluates the noncentral $F$ cumulative distribution function (CDF).

Parameters
- $f$ – A double value representing the argument at which the function is to be evaluated. $f$ must be nonnegative.
- $df1$ – A double value representing the number of numerator degrees of freedom. $df1$ must be positive.
- $df2$ – A double value representing the number of denominator degrees of freedom. $df2$ must be positive.
lambda – A double value representing the noncentrality parameter. lambda must be nonnegative.

**Returns**
A double scalar value representing the probability that a noncentral F random variable takes a value less than or equal to f.

**Remarks**
The noncentral F distribution is a generalization of the F distribution. If X is a noncentral chi-square random variable with noncentrality parameter \( \lambda \) and \( \nu_1 \) degrees of freedom, and Y is a chi-square random variable with \( \nu_2 \) degrees of freedom which is statistically independent of X, then

\[
F = \frac{(X/\nu_1)}{(Y/\nu_2)}
\]

is a noncentral F-distributed random variable whose CDF is given by:

\[
CDF(f, \nu_1, \nu_2, \lambda) = \sum_{j=0}^{\infty} c_j
\]

where:

\[
c_j = \omega_j I_x(\nu_1/2 + j, \nu_1/2)
\]

\[
\omega_j = e^{-\lambda/2} \frac{\lambda}{2^j} \omega_{j-1}
\]

\[
I_x(a, b) = \frac{B_x(a, b)}{B(a, b)}
\]

\[
B_x(a, b) = \int_0^\infty t^{a-1}(1-t)^{b-1}dt = x^n \sum_{j=0}^{\infty} \frac{\Gamma(j+1-b)}{(a+j) \Gamma(1-b) j!} x^j
\]

\[
B(a, b) = B_1(a, b) = \frac{\Gamma(a) \Gamma(b)}{\Gamma(a+b)}
\]

\[
I_x(a+1, b) = I_x(a, b) - T_x(a, b)
\]

\[
T_x(a, b) = \frac{\Gamma(a+b)}{\Gamma(a+1) \Gamma(b)} x^n (1-x)^b = T_x(a-1, b) \frac{a-1+b}{a} x
\]

\[
x = \frac{\nu_1 f}{\nu_2 + \nu_1 f}
\]

and \( \Gamma(\cdot) \) is the gamma function, \( \nu_1 = df1 \), \( \nu_2 = df2 \), \( \lambda = \text{lambda} \), and \( f = f \).

With a noncentrality parameter of zero, the noncentral F distribution is the same as the F distribution.

**NoncentralstudentsT**
static public double NoncentralstudentsT(double t, int idf, double delta)

**Description**
Evaluates the noncentral Student’s t cumulative probability distribution function.
Parameters

- **t** – A double scalar value representing the argument at which the function is to be evaluated.
- **idf** – An int scalar value representing the number of degrees of freedom. This must be positive.
- **delta** – A double scalar value representing the noncentrality parameter.

Returns

A double scalar value representing the probability that a noncentral Student’s *t* random variable takes a value less than or equal to *t*.

Remarks

Method `Cdf.NoncentralstudentsT` evaluates the distribution function *F* of a noncentral *t* random variable with *idf* degrees of freedom and noncentrality parameter *delta*; that is, with *v* = *idf*, *\delta* = *delta*, and *t*₀ = *t*,

\[
F(t_0) = \int_{-\infty}^{t_0} \frac{v^{\nu/2}e^{-\delta^2/2}}{\sqrt{\pi \Gamma(v/2)} (v+x^2)^{(v+1)/2}} \sum_{i=0}^{\infty} \frac{\Gamma((v+i+1)/2)}{i!} \left( \frac{2x^2}{v+x^2} \right)^{i/2} dx
\]

where *Γ(·)* is the gamma function. The value of the distribution function at the point *t*₀ is the probability that the random variable takes a value less than or equal to *t*₀.

The noncentral *t* random variable can be defined by the distribution function above, or alternatively and equivalently, as the ratio of a normal random variable and an independent chi-squared random variable. If *w* has a normal distribution with mean *\delta* and variance equal to one, *u* has an independent chi-squared distribution with *v* degrees of freedom, and

\[
x = w / \sqrt{u/v}
\]

then *x* has a noncentral *t* distribution with *v* degrees of freedom and noncentrality parameter *\delta*.

The distribution function of the noncentral *t* can also be expressed as a double integral involving a normal density function (see, for example, Owen 1962, page 108). The method `Cdf.NoncentralstudentsT` uses the method of Owen (1962, 1965), which uses repeated integration by parts on that alternate expression for the distribution function.
Normal

static public double Normal(double x)

Description
Evaluates the normal (Gaussian) cumulative probability distribution function.

Parameter

x – A double scalar value representing the argument at which the function is to be evaluated.

Returns
A double scalar value representing the probability that a normal variable takes a value less than or equal to x.
Remarks
Method Cdf.Normal evaluates the distribution function, Φ, of a standard normal (Gaussian) random variable, that is,

\[ \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} \, dt \]

The value of the distribution function at the point \( x \) is the probability that the random variable takes a value less than or equal to \( x \).

The standard normal distribution (for which Cdf.Normal is the distribution function) has mean of 0 and variance of 1. The probability that a normal random variable with mean \( \mu \) and variance \( \sigma^2 \) is less than \( y \) is given by Cdf.Normal evaluated at \((y - \mu)/\sigma\).

\( \Phi(x) \) is evaluated by use of the complementary error function, erfc. The relationship is:

\[ \Phi(x) = \text{erfc}\left(-\frac{x}{\sqrt{2}}\right)/2 \]
```csharp
Pareto
static public double Pareto(double x, double xm, double k)

Description
Evaluates the Pareto cumulative probability distribution function.

Parameters
- x – A double scalar value representing the argument at which the function is to be evaluated.
- xm – A double scalar value representing the scale parameter, x_m.
- k – A double scalar value representing the shape parameter, k.

Returns
A double scalar value representing the probability that a Pareto random variable takes a value less than or equal to x.

Remarks
Method Cdf.Pareto evaluates the distribution function, F, of a Pareto random variable with scale parameter x_m and shape parameter k. It is given by

\[ F(x, x_m, k) = 1 - \left(\frac{x_m}{x}\right)^k \]

where \( x_m > 0 \) and \( k > 0 \). The function is only defined for \( x \geq x_m \).

Poisson
static public double Poisson(int k, double theta)

Description
Evaluates the Poisson cumulative probability distribution function.

Parameters
- k – The int argument for which the Poisson distribution function is to be evaluated.
- theta – A double scalar value representing the mean of the Poisson distribution.

Returns
A double scalar value representing the probability that a Poisson random variable takes a value less than or equal to k.

Remarks
Method Cdf.Poisson evaluates the distribution function of a Poisson random variable with parameter theta. theta, which is the mean of the Poisson random variable, must be positive. The probability function (with \( \theta = \text{theta} \)) is

\[ f(x) = e^{-\theta} \frac{\theta^x}{x!} \] for \( x = 0, 1, 2, \ldots \)

The individual terms are calculated from the tails of the distribution to the mode of the distribution and summed. Cdf.Poisson uses the recursive relationship
```
\[ f(x) = f(x) \left( \frac{\theta}{x+1} \right), \text{ for } x = 0, 1, 2, \ldots k - 1 \]

with \( f(0) = e^{-\theta} \).

**Rayleigh**

```java
static public double Rayleigh(double x, double alpha)
```

**Description**

Evaluates the Rayleigh cumulative probability distribution function.

**Parameters**

- `x` – A double scalar value representing the argument at which the function is to be evaluated. It must be non-negative.
- `alpha` – A double scalar value representing the scale parameter.

**Returns**

A double scalar value representing the probability that a Rayleigh random variable takes a value less than or equal to `x`.

**Remarks**

Method `Cdf.Rayleigh` is a special case of Weibull distribution function where the shape parameter `gamma` is 2.0; that is,

\[ F(x) = 1 - e^{-\frac{x^2}{2\alpha}} \]

where `alpha` is the scale parameter.
**StudentsT**

```csharp
static public double StudentsT(double t, double df)
```

**Description**
Evaluates the Student’s t cumulative probability distribution function.

**Parameters**
- `t` – A double scalar value representing the argument at which the function is to be evaluated.
- `df` – A double scalar value representing the number of degrees of freedom. This must be at least one.

**Returns**
A double scalar value representing the probability that a Student’s t random variable takes a value less than or equal to `t`.

---

**Rayleigh Distribution Function**

[Graph showing Rayleigh distribution with different values of alpha]
Remarks

Method Cdf.StudentsT evaluates the distribution function of a Student’s t random variable with df degrees of freedom. If the square of t is greater than or equal to df, the relationship of a t to an f random variable (and subsequently, to a beta random variable) is exploited, and routine Cdf.Beta is used. Otherwise, the method described by Hill (1970) is used. If df is not an integer, if df is greater than 19, or if df is greater than 200, a Cornish-Fisher expansion is used to evaluate the distribution function. If df is less than 20 and |t| is less than 2.0, a trigonometric series (see Abramowitz and Stegun 1964, equations 26.7.3 and 26.7.4, with some rearrangement) is used. For the remaining cases, a series given by Hill (1970) that converges well for large values of t is used.

For greater right tail accuracy, see Cdf.ComplementaryStudentsT (p. 1111).

---

Uniform

static public double Uniform(double x, double aa, double bb)
**Description**
Evaluates the uniform cumulative probability distribution function.

**Parameters**
- \( x \) – A double scalar value representing the argument at which the function is to be evaluated.
- \( aa \) – A double scalar value representing the location parameter.
- \( bb \) – A double scalar value representing the scale parameter.

**Returns**
A double scalar value representing the probability that a Uniform random variable takes a value less than or equal to \( x \).

**Remarks**
Method `Cdf.Uniform` evaluates the distribution function, \( F \), of a uniform random variable with location parameter \( aa \) and scale parameter \( bb \); that is,

\[
f(x) = \begin{cases} 
0 & \text{if } x < aa \\
\frac{x - aa}{bb - aa} & \text{if } aa \leq x \leq bb \\
1 & \text{if } x > bb 
\end{cases}
\]
Weibull

static public double Weibull(double x, double gamma, double alpha)

Description
Evaluates the Weibull cumulative probability distribution function.

Parameters

x – A double scalar value representing the argument at which the function is to be evaluated. It must be non-negative.

gamma – A double scalar value representing the shape parameter, \( \gamma \).

alpha – A double scalar value representing the scale parameter, \( \alpha \).

Returns
A double scalar value representing the probability that a Weibull random variable takes a value less than
or equal to $x$.

**Remarks**

Method `Cdf.Weibull` evaluates the distribution function given by

$$F(x, \gamma, \alpha) = 1 - e^{-(x/\alpha)^\gamma}$$

**Example: The Cumulative Distribution Functions**

Various cumulative distribution functions are exercised. Their use in this example typifies the manner in which other functions in the `Cdf` class would be used.

```csharp
using System;
using Imsl.Stat;

public class CdfEx1
{
    public static void Main(String[] args)
    {
        double x, prob, result;
        int p, q, k, n;
        // Beta
        x = .5;
        p = 12;
        q = 12;
        result = Cdf.Beta(x, p, q);
        Console.Out.WriteLine("Cdf.Beta(.5, 12, 12) is " + result);
        // binomial
        k = 3;
        n = 5;
        prob = .95;
        result = Cdf.Binomial(k, n, prob);
        Console.Out.WriteLine("Cdf.Binomial(3, 5, .95) is " + result);
        // Chi
        x = .15;
        n = 2;
        result = Cdf.Chi(x, n);
        Console.Out.WriteLine("Cdf.Chi(.15, 2) is " + result);
    }
}
```

**Output**

```
Cdf.Beta(.5, 12, 12) is 0.500000000000002
Cdf.Binomial(3, 5, .95) is 0.0225925
Cdf.Chi(.15, 2) is 0.0722565136714471
```
**InvCdf Class**

```java
public class Imsl.Stat.InvCdf
Inverse cumulative probability distribution functions.
```

**See Also**

Cumulative Distribution Function (p. 1137), Probability density function (p. 1149)

**Constructor**

```java
public InvCdf()
```

**Description**


**Methods**

**Beta**

```java
static public double Beta(double p, double pin, double qin)
```

**Description**

Evaluates the inverse of the beta cumulative probability distribution function.

**Parameters**

- `p` – A double, the probability for which the inverse of the beta CDF is to be evaluated.
- `pin` – A double, the first beta distribution parameter.
- `qin` – A double, the second beta distribution parameter.

**Returns**

A double, the probability that a beta random variable takes a value less than or equal to this returned value is `p`. 
Remarks
Method InvCdf.Beta evaluates the inverse distribution function of a beta random variable with parameters \( p = p \) and \( q = q \), that is, with \( P = p \) and \( q = q \), it determines \( x \) (equal to InvCdf.Beta(p, pin, qin)), such that

\[
P = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)} \int_0^x t^{p-1}(1-t)^{q-1} dt
\]

where \( \Gamma(\cdot) \) is the Gamma function. The probability that the random variable takes a value less than or equal to \( x \) is \( P \).

Chi
static public double Chi(double p, double df)

Description
Evaluates the inverse of the chi-squared cumulative probability distribution function.

Parameters
- \( p \) – A double scalar value representing the probability for which the inverse chi-squared function is to be evaluated.
- \( df \) – A double scalar value representing the number of degrees of freedom. This must be at least 0.5.

Returns
A double scalar value. The probability that a chi-squared random variable takes a value less than or equal to this returned value is \( p \).

Remarks
Method InvCdf.Beta evaluates the inverse distribution function of a beta random variable with \( df \) degrees of freedom, that is, with \( P = p \) and \( v = df \), it determines \( x \) (equal to InvCdf.Beta(p, pin, qin)), such that

\[
P = \frac{1}{2^v/2\Gamma(v/2)} \int_0^x e^{-t/v/2}t^{v/2-1} dt
\]

where \( \Gamma(\cdot) \) is the Gamma function. The probability that the random variable takes a value less than or equal to \( x \) is \( P \).

For \( v < 40 \), InvCdf.Beta uses bisection, if \( v \geq 2 \) or \( P > 0.98 \), or regula falsi to find the point at which the chi-squared distribution function is equal to \( P \). The distribution function is evaluated using Cdf.Beta (p. 1106). For 40 \( \leq v < 100 \), a modified Wilson-Hilferty approximation (Abramowitz and Stegun 1964, equation 26.4.18) to the normal distribution is used, and InvCdf.Normal is used to evaluate the inverse of the normal distribution function. For \( v \geq 100 \), the ordinary Wilson-Hilferty approximation (Abramowitz and Stegun 1964, equation 26.4.17) is used.

DiscreteUniform
static public int DiscreteUniform(double p, int n)

Description
Returns the inverse of the discrete uniform cumulative probability distribution function.
Parameters
  p – A double scalar value representing the probability for which the inverse discrete uniform
  function is to be evaluated.
  n – An int scalar value representing the upper limit of the discrete uniform distribution.

Returns
An int scalar value. The probability that a discrete uniform random variable takes a value less than or
equal to this returned value is p.

Exponential
static public double Exponential(double p, double scale)

Description
Evaluates the inverse of the exponential cumulative probability distribution function.

Parameters
  p – A double scalar value representing the probability at which the function is to be evaluated.
  scale – A double scalar value representing the scale parameter.

Returns
A double scalar value. The probability that an exponential random variable takes a value less than or
equal to this returned value is p.

Remarks
Method InvCdf.Exponential evaluates the inverse distribution function of a Gamma random variable
with scale parameter =b and shape parameter a=1.0, that is, it determines x = exponential(p, 1.0)), such
that
\[ P = \frac{1}{\Gamma(a)} \int_0^x e^{-t/b} dt \]
where Γ(·) is the Gamma function. The probability that the random variable takes a value less than or
equal to x is P. See the Cdf.Gamma (p. 1117) remarks for further discussion of the Gamma distribution.
InvCdf.Exponential uses bisection and modified regula falsi to invert the distribution function, which
is evaluated using method Cdf.Gamma.

ExtremeValue
static public double ExtremeValue(double p, double mu, double beta)

Description
Returns the inverse of the extreme value cumulative probability distribution function.

Parameters
  p – A double scalar value representing the probability for which the inverse extreme value
  function is to be evaluated.
  mu – A double scalar value representing the location parameter.
  beta – A double scalar value representing the scale parameter.
Returns
A double scalar value. The probability that an extreme value random variable takes a value less than or equal to this returned value is \( p \).

\[
F \quad \text{static public double } F(\text{double } p, \text{double } dfn, \text{double } dfd)
\]

Description
Returns the inverse of the F cumulative probability distribution function.

Parameters
\( p \) – A double, the probability for which the inverse of the F distribution function is to be evaluated. Argument \( p \) must be in the open interval (0.0, 1.0).

\( dfn \) – A double, the numerator degrees of freedom. It must be positive.

\( dfd \) – A double, the denominator degrees of freedom. It must be positive.

Returns
A double, the probability that an F random variable takes a value less than or equal to this returned value is \( p \).

Remarks
Method InvCdf.F evaluates the inverse distribution function of a Snedecor’s F random variable with \( dfn \) numerator degrees of freedom and \( dfd \) denominator degrees of freedom. The function is evaluated by making a transformation to a beta random variable and then using InvCdf.Beta (p. 1137). If \( X \) is an F variate with \( v_1 \) and \( v_2 \) degrees of freedom and \( Y = v_1X/(v_2 + v_1X) \), then \( Y \) is a beta variate with parameters \( p = v_1/2 \) and \( q = v_2/2 \). If \( P \leq 0.5 \), \( F \) uses this relationship directly, otherwise, it also uses a relationship between \( X \) random variables that can be expressed as follows, using \( f \), which is the \( F \) cumulative distribution function:

\[
F(X, dfn, dfd) = 1 - F(1/X, dfd, dfn)
\]

\[
\text{Gamma} \quad \text{static public double } \text{Gamma(double } p, \text{double } a)
\]

Description
Evaluates the inverse of the Gamma cumulative probability distribution function.

Parameters
\( p \) – A double scalar value representing the probability at which the function is to be evaluated.

\( a \) – A double scalar value representing the shape parameter. This must be positive.

Returns
A double scalar value. The probability that a Gamma random variable takes a value less than or equal to this returned value is \( p \).
Remarks
Method InvCdf.Gamma evaluates the inverse distribution function of a Gamma random variable with shape parameter \( a \), that is, it determines \( x = \text{Gamma}(p,a) \), such that

\[
P = \frac{1}{\Gamma(a)} \int_{0}^{x} e^{-t} t^{a-1} dt
\]

where \( \Gamma(\cdot) \) is the Gamma function. The probability that the random variable takes a value less than or equal to \( x \) is \( P \). See the Cdf.Gamma (p. 1117) remarks for further discussion of the Gamma distribution. InvCdf.Gamma uses bisection and modified regula falsi to invert the distribution function, which is evaluated using method Cdf.Gamma.

Geometric
static public double Geometric(double r, double pin)

Description
Returns the inverse of the discrete geometric cumulative probability distribution function.

Parameters
- \( r \) – A double scalar value representing the probability for which the inverse geometric function is to be evaluated.
- \( pin \) – An int scalar value representing the probability parameter for each independent trial (the probability of success for each independent trial).

Returns
A double scalar value. The probability that a geometric random variable takes a value less than or equal to this returned value is the input probability, \( r \).

Logistic
static public double Logistic(double p, double mu, double s)

Description
Returns the inverse of the logistic cumulative probability distribution function.

Parameters
- \( p \) – A double scalar value representing the probability for which the inverse logistic function is to be evaluated.
- \( mu \) – A double scalar value representing the location parameter, \( \mu \).
- \( s \) – A double scalar value representing the scale parameter.

Returns
A double scalar value. The probability that a logistic random variable takes a value less than or equal to this returned value is \( p \).

LogNormal
static public double LogNormal(double p, double mu, double sigma)
Description
Returns the inverse of the standard LogNormal cumulative probability distribution function.

Parameters
- p – A double scalar value representing the probability for which the inverse LogNormal function is to be evaluated.
- mu – A double scalar value representing the location parameter.
- sigma – A double scalar value representing the shape parameter. sigma must be a positive.

Returns
A double scalar value. The probability that a standard LogNormal random variable takes a value less than or equal to this returned value is p.

NoncentralBeta
static public double NoncentralBeta(double p, double shape1, double shape2, double lambda)

Description
Evaluates the inverse of the noncentral beta cumulative distribution function (CDF).

Parameters
- p – A double scalar value representing the probability for which the inverse of the noncentral beta cumulative distribution function is to be evaluated. p must be non-negative and less than or equal to one.
- shape1 – A double scalar value representing the first shape parameter. shape1 must be positive.
- shape2 – A double scalar value representing the second shape parameter. shape2 must be positive.
- lambda – A double scalar value representing the noncentrality parameter. lambda must be nonnegative.

Returns
A double scalar value representing the inverse of the noncentral beta distribution function evaluated at p. The probability that a noncentral beta random variable takes a value less than or equal to NoncentralBeta is p.

Remarks
If Z is a noncentral chi-square random variable with noncentrality parameter \( \lambda \) and \( 2\alpha_1 \) degrees of freedom, and Y is a chi-square random variable with \( 2\alpha_2 \) degrees of freedom which is statistically independent of Z, then
\[
X = \frac{Z}{Z + Y} = \frac{\alpha_1 F}{\alpha_1 F + \alpha_2}
\]
is a noncentral beta-distributed random variable and
\[
F = \frac{\alpha_2 Z}{\alpha_1 Y} = \frac{\alpha_2 X}{\alpha_1 (1 - X)}
\]
is a noncentral $F$-distributed random variable. The CDF for noncentral beta variable $X$ can thus be simply defined in terms of the noncentral $F$ CDF:

$$CDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda) = CDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda)$$

where $CDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda)$ is the noncentral beta CDF with $x = x$, $\alpha_1 = \text{shape1}$, $\alpha_2 = \text{shape2}$, and noncentrality parameter $\lambda = \text{lambda}$; $CDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda)$ is the noncentral $F$ CDF with argument $f$, numerator and denominator degrees of freedom $2\alpha_1$ and $2\alpha_2$ respectively, and noncentrality parameter $\lambda$; and:

$$f = \frac{\alpha_2 x}{\alpha_1 (1 - x)}; \quad x = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2}$$

(See documentation for class Cdf method NoncentralF for a discussion of how the noncentral $F$ CDF is defined and calculated.)

Method InvCdf.NoncentralBeta evaluates

$$x = CDF^{-1}_{nc\beta}(p, \alpha_1, \alpha_2, \lambda)$$

by first evaluating:

$$f = CDF^{-1}_{ncF}(p, 2\alpha_1, 2\alpha_2, \lambda)$$

and then solving for $x$ using $x = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2}$. (See documentation for InvCdf.NoncentralF (p. 1144) for a discussion of how the inverse noncentral $F$ CDF is calculated.)

Noncentralchi

static public double Noncentralchi(double p, double df, double alam)

Description

Evaluates the inverse of the noncentral chi-squared cumulative probability distribution function.

Parameters

- $p$ – A double scalar value representing the probability for which the inverse noncentral chi-squared distribution function is to be evaluated. $p$ must be in the open interval $(0.0, 1.0)$.
- $df$ – A double scalar value representing the number of degrees of freedom. This must be at least 0.5. but less than or equal to 200,000.
- $alam$ – A double scalar value representing the noncentrality parameter. This must be nonnegative, and $alam + df$ must be less than or equal to 200,000.

Returns

A double scalar value. The probability that a noncentral chi-squared random variable takes a value less than or equal to this returned value is $p$.

Remarks

Method InvCdf.Noncentralchi evaluates the inverse distribution function of a noncentral chi-squared random variable with $df$ degrees of freedom and noncentrality parameter $alam$, that is, with $P = p$, $\nu = df$, and $\lambda = alam$, it determines $c_0 = \text{InvCdf.Noncentralchi}(p, df, alam)$, such that

$$P = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} \int_0^c x^{(\nu+2i)/2-1} e^{-x/2} dx$$
where $\Gamma(\cdot)$ is the Gamma function. The probability that the random variable takes a value less than or equal to $c_0$ is $P$.

Method InvCdf.Noncentralchi uses bisection and modified regula falsi to invert the distribution function, which is evaluated using Cdf.Noncentralchi. See Cdf.Noncentralchi (p. 1123) for an alternative definition of the noncentral chi-squared random variable in terms of normal random variables.

**NoncentralF**

static public double NoncentralF(double p, double dfn, double dfd, double lambda)

**Description**

Evaluates the inverse of the noncentral F cumulative distribution function (CDF).

**Parameters**

- p – A double scalar value representing the probability for which the inverse of the noncentral $F$ cumulative distribution function is to be evaluated. $p$ must be non-negative and less than one.
- dfn – A double scalar value representing the number of numerator degrees of freedom. dfn must be positive.
- dfd – A double scalar value representing the number of denominator degrees of freedom. dfd must be positive.
- lambda – A double scalar value representing the noncentrality parameter. lambda must be non-negative.

**Returns**

A double scalar value representing the inverse of the noncentral $F$ distribution function evaluated at $p$. The probability that a noncentral $F$ random variable takes a value less than or equal to InvCdf.NoncentralF(p, dfn, dfd, lambda) is $p$.

**Remarks**

If $X$ is a noncentral chi-square random variable with noncentrality parameter $\lambda$ and $\nu_1$ degrees of freedom, and $Y$ is a chi-square random variable with $\nu_2$ degrees of freedom which is statistically independent of $X$, then

$$ F = \frac{X}{\nu_1}/(Y/\nu_2) $$

is a noncentral $F$-distributed random variable whose CDF is given by:

$$ CDF(f, \nu_1, \nu_2, \lambda) = \int_0^f PDF(x, \nu_1, \nu_2, \lambda) dx $$

where the probability density function $PDF(x, \nu_1, \nu_2, \lambda)$ is given by:

$$ PDF(x, \nu_1, \nu_2, \lambda) = \Psi \sum_{k=0}^{\infty} \Phi_k $$

$$ \Psi = \frac{e^{-\lambda/2}(\nu_1x)^{\nu_1/2}(\nu_2)^{\nu_2/2}}{x^{(\nu_1x + \nu_2)^{(\nu_1+\nu_2)/2}} \Gamma(\nu_2/2)} $$
\[
\Phi_k = R^k \frac{\Gamma\left(\frac{\nu_1 + \nu_2}{2} + k\right)}{k! \Gamma\left(\frac{\nu_1}{2} + k\right)}
\]

\[
R = \frac{\lambda \nu_1 x}{2(\nu_1 x + \nu_2)}
\]

where \(\Gamma(\cdot)\) is the Gamma function, \(\nu_1 = \text{dfn}, \nu_2 = \text{dfd}, \lambda = \text{lambda}, \) and \(p = CDF(f, \nu_1, \nu_2, \lambda)\) is the probability that \(F \leq f\).

Method InvCdf.NoncentralF evaluates

\[
f = CDF^{-1}(p, \nu_1, \nu_2, \lambda)
\]

Method InvCdf.NoncentralF uses bisection and modified regula falsi search algorithms to invert the distribution function \(CDF(f, \nu_1, \nu_2, \lambda)\), which is evaluated using method Cdf.NoncentralF (p. 1125). For sufficiently small \(p\), an accurate approximation of \(CDF^{-1}(p, \nu_1, \nu_2, \lambda)\) can be used which requires no such inverse search algorithms.

**NoncentralstudentsT**

static public double NoncentralstudentsT(double p, int idf, double delta)

**Description**

Evaluates the inverse of the noncentral Student’s \(t\) cumulative probability distribution function.

**Parameters**

- \(p\) – A double scalar value representing the probability for which the function is to be evaluated.
- \(idf\) – An int scalar value representing the number of degrees of freedom. This must be positive.
- \(delta\) – A double scalar value representing the noncentrality parameter.

**Returns**

A double scalar value. The probability that a noncentral Student’s \(t\) random variable takes a value less than or equal to this returned value is \(p\).

**Remarks**

Method InvCdf.NoncentralstudentsT evaluates the inverse distribution function of a noncentral \(t\) random variable with \(idf\) degrees of freedom and noncentrality parameter \(delta\); that is, with \(P = p\), \(\nu = idf\), \(\delta = delta\), it determines \(t_0 = \text{InvCdf.NoncentralstudentsT}(p, idf, delta)\), such that

\[
P = \int_{-\infty}^{t_0} \frac{\nu^{\nu/2} e^{-\delta^2/2}}{\sqrt{\pi \Gamma(\nu/2)}(\nu + x^2)^{(\nu+1)/2}} \sum_{i=0}^{\infty} \Gamma((\nu + i + 1)/2) \left(\frac{\delta^i}{i!}\right) \left(\frac{2x^2}{\nu + x^2}\right)^i dx
\]

where \(\Gamma(\cdot)\) is the Gamma function. The probability that the random variable takes a value less than or equal to \(t_0\) is \(P\). See Cdf.NoncentralstudentsT (p. 1126) for an alternative definition in terms of normal and chi-squared random variables. The method InvCdf.NoncentralstudentsT uses bisection and modified regula falsi to invert the distribution function, which is evaluated using Cdf.NoncentralstudentsT.

**Normal**

static public double Normal(double p)
Description
Evaluates the inverse of the normal (Gaussian) cumulative probability distribution function.

Parameter
\[ p \] – A double scalar value representing the probability at which the function is to be evaluated.

Returns
A double scalar value. The probability that a standard normal random variable takes a value less than or equal to this returned value is \( p \).

Remarks
Method InvCdf.Normal evaluates the inverse of the distribution function, \( \Phi \), of a standard normal (Gaussian) random variable, that is, \( \text{InvCdf.Normal}(p) = \Phi^{-1}(p) \), where
\[
\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt
\]
The value of the distribution function at the point \( x \) is the probability that the random variable takes a value less than or equal to \( x \). The standard normal distribution has a mean of 0 and a variance of 1.

Pareto
static public double Pareto(double p, double xm, double k)

Description
Returns the inverse of the Pareto cumulative probability density function.

Parameters
\[ p \] – A double scalar value representing the probability for which the inverse Pareto function is to be evaluated.
\[ xm \] – A double scalar value representing the scale parameter, \( x_m \).
\[ k \] – A double scalar value representing the shape parameter.

Returns
A double scalar value. The probability that a Pareto random variable takes on a value less than or equal to this returned value is \( p \).

Rayleigh
static public double Rayleigh(double p, double alpha)

Description
Returns the inverse of the Rayleigh cumulative probability distribution function.

Parameters
\[ p \] – A double scalar value representing the probability for which the inverse Rayleigh function is to be evaluated.
\[ alpha \] – A double scalar value representing the scale parameter.
Returns
A double scalar value. The probability that a Rayleigh random variable takes a value less than or equal to this returned value is \( p \).

**StudentsT**
static public double StudentsT(double p, double df)

**Description**
Returns the inverse of the Student’s t cumulative probability distribution function.

**Parameters**
- \( p \) – A double scalar value representing the probability for which the inverse Student’s \( t \) function is to be evaluated.
- \( df \) – A double scalar value representing the number of degrees of freedom. This must be at least one.

**Returns**
A double scalar value. The probability that a Student’s \( t \) random variable takes a value less than or equal to this returned value is \( p \).

**Remarks**
InvCdf.StudentsT evaluates the inverse distribution function of a Student’s \( t \) random variable with \( df \) degrees of freedom. Let \( v = df \). If \( v \) equals 1 or 2, the inverse can be obtained in closed form, if \( v \) is between 1 and 2, the relationship of a \( t \) to a beta random variable is exploited and InvCdf.Beta (p. 1137) is used to evaluate the inverse; otherwise the algorithm of Hill (1970) is used. For small values of \( v \) greater than 2, Hill’s algorithm inverts an integrated expansion in \( 1/(1+t^2/v) \) of the \( t \) density. For larger values, an asymptotic inverse Cornish-Fisher type expansion about normal deviates is used.

**Uniform**
static public double Uniform(double p, double aa, double bb)

**Description**
Returns the inverse of the uniform cumulative probability distribution function.

**Parameters**
- \( p \) – A double scalar value representing the probability for which the inverse uniform function is to be evaluated.
- \( aa \) – A double scalar value representing the minimum value.
- \( bb \) – A double scalar value representing the maximum value.

**Returns**
A double scalar value. The probability that a uniform random variable takes a value less than or equal to this returned value is \( p \).

**Weibull**
static public double Weibull(double p, double gamma, double alpha)
Description
Returns the inverse of the Weibull cumulative probability distribution function.

Parameters
- p – A double scalar value representing the probability for which the inverse Weibull function is to be evaluated.
- gamma – A double scalar value representing the shape parameter.
- alpha – A double scalar value representing the scale parameter.

Returns
A double scalar value. The probability that a Weibull random variable takes a value less than or equal to this returned value is p.

Example: The Inverse Cumulative Distribution Functions
Examples of the inverse cumulative distribution functions are exercised. Their use in this example typifies the manner in which other functions in the InvCdf class would be used.

```csharp
using System;
using InvCdf = Imsl.Stat.InvCdf;

public class InvCdfEx1
{
    public static void Main(string[] args)
    {
        // Inverse Beta
        double x = 0.5;
        double pin = 12.0;
        double qin = 12.0;
        double result = InvCdf.Beta(x, pin, qin);
        Console.Out.WriteLine("InvCdf.Beta(0.5, 12., 12.) is {0,5:0.0000}", result);

        // Inverse Chi
        double prob = 0.99;
        int n = 2;
        result = InvCdf.Chi(prob, n);
        Console.Out.WriteLine("InvCdf.Chi(0.99, 2) is {0,5:0.0000}", result);
    }
}
```

Output
InvCdf.Beta(0.5, 12., 12.) is 0.5000
InvCdf.Chi(0.99, 2) is 9.2103
Pdf Class

public class Imsl.Stat.Pdf
Probability density functions.

See Also
Cumulative distribution function (p. 1101), Inverse Cumulative Distribution Function (p. 1137)

Methods

Beta
static public double Beta(double x, double pin, double qin)

Description
Evaluates the beta probability density function.

Parameters
- x – A double, the argument at which the function is to be evaluated.
- pin – A double, the first beta distribution parameter.
- qin – A double, the second beta distribution parameter.

Returns
A double, the value of the probability density function at x.

Binomial
static public double Binomial(int k, int n, double pin)

Description
Evaluates the binomial probability density function.

Parameters
- k – The int argument for which the binomial distribution function is to be evaluated.
- n – The int number of Bernoulli trials.
- pin – A double scalar value representing the probability of success on each independent trial.

Returns
A double scalar value representing the probability that a binomial random variable takes a value equal to k.
Remarks
Method Pdf.Binomial evaluates the probability that a binomial random variable with parameters \( n \) and \( p \) with \( p = \text{pin} \) takes on the value \( k \). It does this by computing probabilities of the random variable taking on the values in its range less than (or the values greater than) \( k \). These probabilities are computed by the recursive relationship

\[
\Pr(X = j) = \frac{(n + 1 - j)p}{j(1 - p)} \Pr(X = j - 1)
\]

To avoid the possibility of underflow, the probabilities are computed forward from 0, if \( k \) is not greater than \( n \times p \), and are computed backward from \( n \), otherwise. The smallest positive machine number, \( \varepsilon \), is used as the starting value for computing the probabilities, which are rescaled by \((1 - p)^n \varepsilon \) if forward computation is performed and by \( p^n \varepsilon \) if backward computation is done.

For the special case of \( p = 0 \), Pdf.Binomial is set to 0 if \( k \) is greater than 0 and to 1 otherwise; and for the case \( p = 1 \), Pdf.Binomial is set to 0 if \( k \) is less than \( n \) and to 1 otherwise.
Chi
static public double Chi(double chsq, double df)

Description
Evaluates the chi-squared probability density function

Parameters
- chsq – A double scalar value representing the argument at which the function is to be evaluated.
- df – A double scalar value representing the number of degrees of freedom. df must be positive.

Returns
A double scalar value, the value of the probability density function at chsq.

DiscreteUniform
static public double DiscreteUniform(int x, int n)

Probability Distribution Functions and Inverses
Description
Evaluates the discrete uniform probability density function.

Parameters
- x – An int argument for which the discrete uniform probability density function is to be evaluated.
- x should be a value between the lower limit 0 and upper limit n.
- n – An int scalar value representing the upper limit of the discrete uniform distribution.

Returns
A double scalar value representing the probability that a discrete uniform random variable takes a value equal to x.

Exponential
static public double Exponential(double x, double scale)

Description
Evaluates the exponential probability density function.

Parameters
- x – A double scalar value representing the argument at which the function is to be evaluated.
- scale – A double scalar value representing the scale parameter.

Returns
A double scalar value, the value of the probability density function at x.

ExtremeValue
static public double ExtremeValue(double x, double mu, double beta)

Description
Evaluates the extreme value probability density function.

Parameters
- x – A double scalar value representing the argument at which the function is to be evaluated.
- mu – A double scalar value representing the location parameter.
- beta – A double scalar value representing the scale parameter.

Returns
A double scalar value representing the probability density function at x.

F
static public double F(double x, double dfn, double dfd)

Description
Evaluates the F probability density function.
Parameters

- \(x\) – A double, the argument at which the function is to be evaluated.
- \(dfn\) – A double, the numerator degrees of freedom. It must be positive.
- \(dfd\) – A double, the denominator degrees of freedom. It must be positive.

Returns

A double, the value of the probability density function at \(x\).

Remarks

The probability density function of the F distribution is

\[
f(x, dfn, dfd) = \frac{\Gamma\left(\frac{v_1 + v_2}{2}\right)\left(\frac{v_1}{v_2}\right)\left(\frac{x}{v_2}\right)^{v_1/2}}{\Gamma\left(\frac{v_1}{2}\right)\Gamma\left(\frac{v_2}{2}\right)\left(1 + \frac{v_1 x}{v_2}\right)^{\frac{v_1 + v_2}{2}}}
\]

where \(v_1\) and \(v_2\) are the shape parameters \(dfn\) and \(dfd\) and \(\Gamma\) is the gamma function,

\[
\Gamma(a) = \int_0^\infty t^{a-1} e^{-t} dt
\]

**Gamma**

```csharp
static public double Gamma(double x, double a, double b)
```

Description

Evaluates the gamma probability density function.

Parameters

- \(x\) – A double scalar value representing the argument at which the function is to be evaluated.
- \(a\) – A double scalar value representing the shape parameter. This must be positive.
- \(b\) – A double scalar value representing the scale parameter. This must be positive.

Returns

A double scalar value, the probability density function at \(x\).

Remarks

The probability density function of the gamma distribution is

\[
f(x; a, b) = x^{a-1} \frac{1}{b^a \Gamma(a)} e^{-x/b}
\]

where \(a\) is the shape parameter and \(b\) is the scale parameter.

**Geometric**

```csharp
static public double Geometric(int x, double pin)
```

Description

Evaluates the discrete geometric probability density function.
Parameters
x – The int argument for which the geometric probability function is to be evaluated.

pin – A double scalar value representing the probability parameter of the geometric distribution
(the probability of success for each independent trial).

Returns
A double scalar value representing the probability that a geometric random variable takes a value equal
to x.

Remarks
Method Pdf.Geometric evaluates the geometric distribution for the number of trials before the first
success.

Hypergeometric
static public double Hypergeometric(int k, int sampleSize, int defectivesInLot, int lotSize)

Description
Evaluates the hypergeometric probability density function.

Parameters
k – An int, the argument at which the function is to be evaluated.
sampleSize – An int, the sample size, n.
defectivesInLot – An int, the number of defectives in the lot, m.
lotSize – An int, the lot size, l.

Returns
A double, the probability that a hypergeometric random variable takes on a value equal to k.

Remarks
Method Pdf.Hypergeometric evaluates the probability density function of a hypergeometric random
variable with parameters n, l, and m. The hypergeometric random variable X can be thought of as the
number of items of a given type in a random sample of size n that is drawn without replacement from a
population of size l containing m items of this type. The probability density function is:

$$\Pr(X = k) = \binom{m}{k} \binom{l-m}{n-k} \frac{(l-n+k)}{(l-n)}$$

for $$k = i, i + 1, i + 2, \ldots, \min(n, m)$$

where $$i = \max(0, n - l + m)$$. Pdf.Hypergeometric evaluates the expression using log gamma
functions.

Logistic
static public double Logistic(double x, double mu, double s)

Description
Evaluates the logistic probability density function.
Parameters

- \( x \) – A double scalar value representing the argument at which the function is to be evaluated.
- \( \mu \) – A double scalar value representing the location parameter.
- \( s \) – A double scalar value representing the scale parameter.

Returns

A double scalar value representing the probability density function at \( x \).

Remarks

The probability density function of the logistic distribution is

\[
f(x, \mu, s) = \frac{e^{-(x-\mu)/s}}{s \left(1 + e^{-(x-\mu)/s}\right)^2}
\]

where \( \mu \) is the location parameter and the scale parameter \( s > 0 \).

LogNormal

static public double LogNormal(double x, double mu, double sigma)

Description

Evaluates the standard lognormal probability density function.

Parameters

- \( x \) – A double scalar value representing the argument at which the function is to be evaluated.
- \( \mu \) – A double scalar value representing the location parameter.
- \( \sigma \) – A double scalar value representing the shape parameter. \( \sigma \) must be a positive.

Returns

A double scalar value representing the probability density function at \( x \).

Remarks

\[
F(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}
\]

NoncentralBeta

static public double NoncentralBeta(double x, double shape1, double shape2, double lambda)

Description

Evaluates the noncentral beta probability density function.
**Parameters**

- \( x \) – A double scalar value representing the argument at which the function is to be evaluated. \( x \) must be nonnegative and less than or equal to 1.
- \( \text{shape1} \) – A double scalar value representing the first shape parameter. \( \text{shape1} \) must be positive.
- \( \text{shape2} \) – A double scalar value representing the second shape parameter. \( \text{shape2} \) must be positive.
- \( \lambda \) – A double scalar value representing the noncentrality parameter. \( \lambda \) must be nonnegative.

**Returns**

A double scalar value representing the probability density associated with a noncentral beta random variable with value \( x \).

**Remarks**

The noncentral beta distribution is a generalization of the beta distribution. If \( Z \) is a noncentral chi-square random variable with noncentrality parameter \( \lambda \) and \( 2\alpha_1 \) degrees of freedom, and \( Y \) is a chi-square random variable with \( 2\alpha_2 \) degrees of freedom which is statistically independent of \( Z \), then

\[
X = \frac{Z}{Z + Y} = \frac{\alpha_1 F}{\alpha_1 F + \alpha_2}
\]

is a noncentral beta-distributed random variable and

\[
F = \frac{\alpha_2 Z}{\alpha_1 Y} = \frac{\alpha_2 X}{\alpha_1 (1 - X)}
\]

is a noncentral \( F \)-distributed random variable. The PDF for noncentral beta variable \( X \) can thus be simply defined in terms of the noncentral \( F \) PDF:

\[
PDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda) = PDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda) \frac{df}{dx}
\]

where \( PDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda) \) is the noncentral beta PDF with \( x = x \), \( \alpha_1 = \text{shape1} \), \( \alpha_2 = \text{shape2} \), and noncentrality parameter \( \lambda = \lambda \); \( PDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda) \) is the noncentral \( F \) PDF with argument \( f \), numerator and denominator degrees of freedom \( 2\alpha_1 \) and \( 2\alpha_2 \) respectively, noncentrality parameter \( \lambda \),

\[
f = \frac{\alpha_2 x}{\alpha_1 (1 - x)},
\]

\[
x = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2},
\]

and

\[
\frac{df}{dx} = \frac{(\alpha_1 f + \alpha_2)^2}{\alpha_1 \alpha_2} = \frac{\alpha_2}{\alpha_1 (1 - x)^2}.
\]
With a noncentrality parameter of zero, the noncentral beta distribution is the same as the beta distribution.

**NoncentralChi**

```java
static public double NoncentralChi(double chsq, double df, double alam)
```

**Description**

Evaluates the noncentral chi-squared probability density function.

**Parameters**

- `chsq` – A `double` scalar value at which the function is to be evaluated. `chsq` must be nonnegative.
- `df` – A `double` scalar value representing the number of degrees of freedom. `df` must be positive.
- `alam` – A `double` scalar value representing the noncentrality parameter. `alam` must be nonnegative.

**Returns**

A `double` scalar value representing the probability density associated with a noncentral chi-squared random variable with value `chsq`.

**Remarks**

The noncentral chi-squared distribution is a generalization of the chi-squared distribution. If \( \{X_i\} \) are \( k \) independent, normally distributed random variables with means \( \mu_i \) and variances \( \sigma_i^2 \), then the random variable

\[
X = \sum_{i=1}^{k} \left( \frac{X_i}{\sigma_i} \right)^2
\]

is distributed according to the noncentral chi-squared distribution. The noncentral chi-squared distribution has two parameters, \( k \) which specifies the number of degrees of freedom (i.e. the number of \( X_i \)), and \( \lambda \) which is related to the mean of the random variables \( X_i \) by

\[
\lambda = \sum_{i=1}^{k} \left( \frac{\mu_i}{\sigma_i} \right)^2
\]

The noncentral chi-squared distribution is equivalent to a (central) chi-squared distribution with \( k + 2i \) degrees of freedom, where \( i \) is the value of a Poisson distributed random variable with parameter \( \lambda / 2 \). Thus, the probability density function is given by:

\[
F(x, k, \lambda) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2}(\lambda/2)^i}{i!} f(x, k + 2i)
\]

where the (central) chi-squared Pdf \( f(x, k) \) is given by:
\[ f(x, k) = \frac{(x/2)^{k/2} e^{-x/2}}{x \Gamma(k/2)} \quad \text{for } x > 0, \text{ else } 0 \]

where \( \Gamma(\cdot) \) is the gamma function. The above representation of \( F(x, k, \lambda) \) can be shown to be equivalent to the representation:

\[ F(x, k, \lambda) = e^{-\left(\lambda + x\right)/2} \left(\frac{x}{2}\right)^{k/2} \sum_{i=0}^{\infty} \frac{\phi_i}{x^i \Gamma(k/2 + i)} \]

\[ \phi_i = \frac{\left(\frac{\lambda x}{4}\right)^i}{i! \Gamma(k/2 + i)} \]

Method Pdf.NoncentralChi evaluates the probability density function, \( F(x, k, \lambda) \), of a noncentral chi-squared random variable with \( df \) degrees of freedom and noncentrality parameter \( \lambda \), corresponding to \( k = df, \lambda = \lambda \), and \( x = \text{chsq} \).

With a noncentrality parameter of zero, the noncentral chi-squared distribution is the same as the central chi-squared distribution.

**NoncentralF**

static public double NoncentralF(double f, double df1, double df2, double lambda)

**Description**
Evaluates the noncentral \( F \) probability density function.

**Parameters**
- \( f \) – A double value representing the argument at which the function is to be evaluated. \( f \) must be nonnegative.
- \( df1 \) – A double value representing the number of numerator degrees of freedom. \( df1 \) must be positive.
- \( df2 \) – A double value representing the number of denominator degrees of freedom. \( df2 \) must be positive.
- \( \lambda \) – A double value representing the noncentrality parameter. \( \lambda \) must be nonnegative.

**Returns**
A double value representing the probability density associated with a noncentral \( F \) random variable with value \( f \).
Remarks

The noncentral \( F \) distribution is a generalization of the \( F \) distribution. If \( x \) is a noncentral chi-square random variable with noncentrality parameter \( \lambda \) and \( v_1 \) degrees of freedom, and \( y \) is a chi-square random variable with \( v_2 \) degrees of freedom which is statistically independent of \( X \), then

\[
F = \frac{x / v_1}{y / v_2}
\]

is a noncentral \( F \)-distributed random variable whose PDF is given by:

\[
PDF(f, v_1, v_2, \lambda) = \Psi \sum_{k=0}^{\infty} \Phi_k
\]

where

\[
\Psi = \frac{e^{-\lambda/2} (v_1 f)^{v_1/2} (v_2)^{v_2/2}}{f (v_1 f + v_2)^{(v_1+v_2)/2} \Gamma(v_2/2)}
\]

\[
\Phi_k = \frac{R^k \Gamma(v_1 f + v_2)}{k! \Gamma(v_2/2 + k)}
\]

\[
R = \frac{\lambda v_1 f}{2(v_1 f + v_2)}
\]

where \( \Gamma(\cdot) \) is the gamma function, \( v_1 = df_1 \), \( v_2 = df_2 \), \( \lambda = \text{lambda} \), and \( f = f \).

With a noncentrality parameter of zero, the noncentral \( F \) distribution is the same as the \( F \) distribution.

The efficiency of the calculation of the above series is enhanced by:

1. calculating each term \( \Phi_k \) in the series recursively in terms of either the term \( \Phi_{k-1} \) preceding it or the term \( \Phi_{k+1} \) following it, and
2. initializing the sum with the largest series term and adding the subsequent terms in order of decreasing magnitude.

Special cases:

For \( R = \lambda \) and \( f = 0 \):

\[
PDF(f, v_1, v_2, \lambda) = \Psi \Phi_0 = \frac{\Gamma([v_1 + v_2]/2)}{\Gamma(v_1/2)}
\]

For \( \lambda = 0 \):

\[
PDF(f, v_1, v_2, \lambda) = \frac{(v_1 f)^{v_1/2} (v_2)^{v_2/2} \Gamma([v_1 + v_2]/2)}{f (v_1 f + v_2)^{(v_1+v_2)/2} \Gamma(v_1/2) \Gamma(v_2/2)}
\]

For \( f = 0 \):

\[
PDF(f, v_1, v_2, \lambda) = \frac{e^{-\lambda/2} f^{v_1/2} - 1 (v_1/v_2)^{v_1/2} \Gamma([v_1 + v_2]/2)}{\Gamma(v_1/2) \Gamma(v_2/2)}
\]

\[
PDF(f, v_1, v_2, \lambda) = \begin{cases} 
0 & \text{if } v_1 > 2; \\
e^{-\lambda/2} & \text{if } v_1 = 2; \\
\infty & \text{if } v_1 < 2
\end{cases}
\]

**NoncentralStudentsT**

```java
static public double NoncentralStudentsT(double t, double df, double delta)
```

Probability Distribution Functions and Inverses
Description
Evaluates the noncentral Student’s \( t \) probability density function.

Parameters
- \( t \) – A double value representing the argument at which the function is to be evaluated.
- \( df \) – A double value representing the number of degrees of freedom. \( df \) must be positive.
- \( \delta \) – A double value representing the noncentrality parameter.

Returns
A double value representing the probability density associated with a noncentral Student’s \( t \) random variable with value \( t \).

Remarks
The noncentral Student’s \( t \)-distribution is a generalization of the Student’s \( t \)-distribution. If \( w \) is a normally distributed random variable with unit variance and mean \( \delta \) and \( u \) is a chi-square random variable with \( \nu \) degrees of freedom that is statistically independent of \( w \), then

\[
T = \frac{w}{\sqrt{u/\nu}}
\]

is a noncentral \( t \)-distributed random variable with \( \nu \) degrees of freedom and noncentrality parameter \( \delta \), that is, with \( \nu = df \), and \( \delta = \delta \). The probability density function for the noncentral \( t \)-distribution is:

\[
f(t, \nu, \delta) = \frac{\nu^{\nu/2} e^{-\delta^2/2}}{\sqrt{\pi} \Gamma(\nu/2) (\nu + t^2)^{(\nu+1)/2}} \sum_{i=0}^{\infty} \Phi_i
\]

where

\[
\Phi_i = \frac{\Gamma((\nu+i+1)/2)}{i!} \left[ \delta t \right]^i \left( \frac{2}{\nu + t^2} \right)^{i/2}
\]

and \( t = t \).

For noncentrality parameter \( \delta = 0 \), the PDF reduces to the (central) Student’s \( t \) PDF:

\[
f(t, \nu, 0) = \frac{\Gamma((\nu+1)/2)}{\sqrt{\nu \pi} \Gamma(\nu/2)} \left( 1 + \frac{t^2}{\nu} \right)^{-(\nu+1)/2}
\]

and, for \( t = 0 \), the Pdf becomes:

\[
f(0, \nu, \delta) = \frac{\Gamma((\nu+1)/2) e^{-\delta^2/2}}{\sqrt{\nu \pi} \Gamma(\nu/2)}
\]

Normal
static public double Normal(double x, double mean, double stdev)
Description
Evaluates the normal (Gaussian) probability density function.

Parameters
- x – A double scalar value representing the argument at which the function is to be evaluated.
- mean – A double scalar value containing the mean.
- stddev – A double scalar value containing the standard deviation.

Returns
A double containing the value of the probability density function at x

Remarks
The probability density function for a normal distribution is given by
\[ \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]
where \( \mu \) and \( \sigma \) are the conditional mean and standard deviation.

Pareto
static public double Pareto(double x, double xm, double k)

Description
Evaluates the Pareto probability density function.

Parameters
- x – A double scalar value representing the argument at which the function is to be evaluated.
- xm – A double scalar value representing the scale parameter, \( x_m \).
- k – A double scalar value representing the shape parameter.

Returns
A double scalar value representing the probability density function at x.

Remarks
The probability density function of the Pareto distribution is
\[ f(x, x_m, k) = 1 - \frac{k x_m^k}{x^{k+1}} \]
where the scale parameter \( x_m > 0 \) and the shape parameter \( k > 0 \). The function is only defined for \( x \geq x_m \)

Poisson
static public double Poisson(int k, double theta)

Description
Evaluates the Poisson probability density function.
Parameters

- \( k \) – An int scalar for which the Poisson probability function is to be evaluated.
- \( \theta \) – A double scalar value representing the mean of the Poisson distribution.

Returns

A double scalar value representing the probability that a Poisson random variable takes a value equal to \( k \).

Remarks

Method `Pdf.Poisson` evaluates the probability density function of a Poisson random variable with parameter \( \theta \). \( \theta \), which is the mean of the Poisson random variable, must be positive. The probability function (with \( \theta = \theta \)) is

\[
f(x) = e^{-\theta} \frac{\theta^k}{k!}, \quad \text{for } k = 0, 1, 2, \ldots
\]

`Pdf.Poisson` evaluates this function directly, taking logarithms and using the log gamma function.
Rayleigh

static public double Rayleigh(double x, double alpha)

Description
Evaluates the Rayleigh probability density function.

Parameters

- x – A double scalar value representing the argument at which the function is to be evaluated. It must be non-negative.
- alpha – A double scalar value representing the scale parameter.

Returns
A double scalar value representing the probability density function at x.
Weibull

static public double Weibull(double x, double gamma, double alpha)

Description
Evaluates the Weibull probability density function.

Parameters
- x – A double scalar value representing the argument at which the function is to be evaluated. It must be non-negative.
- gamma – A double scalar value representing the shape parameter.
- alpha – A double scalar value representing the scale parameter.

Returns
A double scalar value, the probability density function at x.

Example: The Probability Distribution Functions
Examples of the probability distribution functions are exercised. Their use in this example typifies the manner in which other functions in the Pdf class would be used.

```csharp
using System;

public class PdfEx1
{
    public static void Main(String[] args)
    {

        // Beta
        double x = .5;
        double pin = 12.0;
        double qin = 12.0;
        double result = Pdf.Beta(x, pin, qin);
        Console.Out.WriteLine("Pdf.Beta(.5, 12., 12.) is {0,5:0.0000}", result);

        // binomial
        int k = 3;
        int n = 5;
        double prob = .95;
        result = Pdf.Binomial(k, n, prob);
        Console.Out.WriteLine("Pdf.Binomial(3, 5, .95) is " + result);

        // Chi
        prob = .99;
        n = 2;
        result = Pdf.Chi(prob, n);
        Console.Out.WriteLine("Pdf.Chi(.99, 2) is {0,5:0.0000}", result);
    }
}
```
Output
Pdf.Beta(.5, 12., 12.) is 3.8683
Pdf.Binomial(3, 5, .95) is 0.021434375
Pdf.Chi(.99, 2) is 0.3048

ICdfFunction Interface

public interface Imsl.Stat.ICdfFunction

Interface for the user-supplied cumulative distribution function to be used by InverseCdf and ChiSquaredTest.

Method

CdfFunction
abstract public double CdfFunction(double p)

Description
User-supplied cumulative distribution function to be used by InverseCdf.

Parameter
 p – A double scalar value representing the point at which the inverse CDF is desired.

Returns
A double scalar value representing the probability that a random variable for this CDF takes a value less than or equal to this value is p.

InverseCdf Class

public class Imsl.Stat.InverseCdf

Inverse of user-supplied cumulative distribution function.

Class InverseCdf evaluates the inverse of a continuous, strictly monotone function. Its most obvious use is in evaluating inverses of continuous distribution functions that can be defined by a user-supplied function, which implements the ICdfFunction interface. The inverse is computed using regula falsi and/or bisection, possibly with the Illinois modification (see Dahlquist and Bjorck 1974). A maximum of 100 iterations are performed.
Property

Tolerance
public double Tolerance {get; set; }

Description
The tolerance to be used as the convergence criterion.

Property Value
A double scalar value representing the convergence criterion.

Remarks
When the relative change from one iteration to the next is less than tolerance, convergence is assumed. The default value for tolerance is 0.0001.

Constructor

InverseCdf
public InverseCdf(Imsl.Stat.ICdfFunction cdf)

Description
Constructor for the inverse of a user-supplied cumulative distribution function.

Parameter

cdf – A ICdfFunction object that contains the user-supplied function to be inverted.

Remarks
The cdf function must be continuous and strictly monotone.

Method

Eval
public double Eval(double p, double guess)

Description
Evaluates the inverse CDF function.

Parameters

p – A double scalar value representing the point at which the inverse CDF is desired.

guess – A double scalar value representing an initial estimate of the inverse at p.
Returns

A double scalar value representing the inverse of the CDF at the point \( p \).

Remarks

\( \text{Cdf} \left( \text{InverseCdf} \right) \) is “close” to \( p \).

Exception

\text{Imsl.Stat.DidNotConvergeException} is thrown if the iteration to find the inverse of the CDF did not converge.

Example: Inverse of a User-Supplied Cumulative Distribution Function

In this example, \( \text{InverseCdf} \) is used to compute the point such that the probability is 0.9 that a standard normal random variable is less than or equal to the computed point.

```csharp
using System;
using Imsl.Stat;

public class InverseCdfEx1 : ICdfFunction
{
    public double CdfFunction(double x)
    {
        return Cdf.Normal(x);
    }

    public static void Main(String[] args)
    {
        double p = 0.9;
        ICdfFunction normal = new InverseCdfEx1();
        InverseCdf inv = new InverseCdf(normal);
        inv.Tolerance = 1.0e-10;
        double x1 = inv.Eval(p, 0.0);
        Console.Out.WriteLine("The 90th percentile of a standard normal is " + x1);
    }
}
```

Output

The 90th percentile of a standard normal is 1.2815515655446

---

**IDistribution Interface**

```csharp
public interface Imsl.Stat.IDistribution

Public interface for the user-supplied distribution function.
```

---

**Probability Distribution Functions and Inverses**

---

**IDistribution • 1167**
The purpose of this interface is to fit the probability distribution to a given set of data and return the probability density at each value of the given set of data.

**Method**

**Eval**

abstract public double[] Eval(double[] xData)

**Description**

Evaluation method to fit the user-supplied probability density function to input data.

**Parameter**

xData – A double array representing the points at which the probability density function is to be evaluated.

**Returns**

A double array representing the probability density at each value of xData.

**IProbabilityDistribution Interface**


Public interface for a user-supplied probability distribution.

The purpose of this interface is to evaluate the probability density of a given set of data by either fitting the probability density function to the data or by evaluating the probability density function with supplied parameters. Both Eval methods return the probability density at each value of the given set of data. After the probability distribution is fitted to the data, the GetParameters method can be used to return the distribution parameters used to fit the probability density function to the data.

The DataMining package class NaiveBayesClassifier uses an implementation of IProbabilityDistribution to train continuous data.

**See Also**

Naive Bayes Example 3
Methods

Eval
abstract public double[] Eval(double[] xData, object[] parameters)

Description
Evaluates the user-supplied probability density of each value in xData using the supplied probability distribution parameters.

Parameters
  xData – A double array containing the points at which the probability density function is to be evaluated.
  parameters – An Object array containing the probability distribution parameters to be used in evaluating xData. See method GetParameters.

Returns
A double array representing the probability density of each value of xData.

Eval
abstract public double Eval(double xData, object[] parameters)

Description
Evaluation method for the user-supplied distribution function and parameters. Evaluates the user-supplied probability density at xData using the supplied probability distribution parameters.

Parameters
  xData – A double scalar value containing the point the distribution function is to evaluate.
  parameters – An Object array containing the probability distribution parameters to be used in evaluating xData. See method GetParameters.

Returns
A double scalar value representing the probability density at xData.

GetParameters
abstract public object[] GetParameters()

Description
Returns the current parameters of the probability density function.

Returns
An Object array containing the parameters resulting from the last invocation of the (IDistribution) Eval method with the following signature, double[] Eval(double[] xData). This Object array can be used as input to the Eval methods that require an Object array of distribution parameters as input.
NormalDistribution Class


Evaluates the normal (Gaussian) probability density for a given set of data.

NormalDistribution evaluates the normal probability density of a given set of data, xData. If parameters are not supplied, the Eval method fits the normal probability density function to the data by first calculating the mean and standard deviation of xData. The normal probability density function is defined as:

\[ f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]

where \( \mu \) and \( \sigma \) are the mean and standard deviation.

The DataMining package class NaiveBayesClassifier uses NormalDistribution as the default method to train continuous data.

See Also

Naive Bayes Example 1

Properties

Mean

virtual public double Mean {get; }

Description

Returns the population mean of xData.

Property Value

A double representing the population mean of xData.

StandardDeviation

virtual public double StandardDeviation {get; }

Description

Returns the population standard deviation.

Property Value

A double representing the population standard deviation of xData.
Constructor

**NormalDistribution**

public NormalDistribution()

**Description**


Methods

**Eval**

virtual public double[] Eval(double[] xData)

**Description**

Fits a normal (Gaussian) probability distribution to xData and returns the probability density at each value.

**Parameter**

xData – A double array representing the points at which the normal probability distribution function is to be evaluated.

**Returns**

A double array representing the normal probability density at each value in xData.

**Eval**

virtual public double[] Eval(double[] xData, object[] parameters)

**Description**

Evaluates a normal (Gaussian) probability distribution with the given parameters at each point in xData and returns the probability density at each value.

**Parameters**

xData – A double array representing the points at which the normal probability distribution function is to be evaluated.

parameters – An Object array representing the parameters used to evaluate the normal probability density function, see method GetParameters.

**Returns**

A double array representing the normal probability density of each value in xData.

**Eval**

virtual public double Eval(double xData, object[] parameters)
Description
Evaluates a normal (Gaussian) probability density at a given point xData.

Parameters
- **xData** – A double containing the point at which the normal probability density function is to be evaluated.
- **parameters** – An Object array representing the parameters used to evaluate the normal probability density, see method GetParameters.

Returns
A double representing the normal probability density at xData.

GetParameters
virtual public object[] GetParameters()

Description
Returns the current parameters of the normal probability density function.

Returns
An Object array containing the parameters resulting from the last invocation of the (IDistribution) Eval method with the following signature, double[] Eval(double[] xData). This Object array can be used as input to the Eval methods that require an Object array of distribution parameters as input.

GammaDistribution Class


Evaluates a gamma probability density for a given set of data.

GammaDistribution evaluates the gamma density of a given set of data, xData. If parameters are not supplied, the Eval method fits the gamma probability density function to the data by first calculating the shape and scale parameters using an MLE technique for a best fit. The gamma probability density function is defined as:

\[ f(x) = x^{a-1} \frac{e^{-\frac{x}{b}}}{b^a \Gamma(a)}, \quad x > 0, \quad a > 0 \quad \text{and} \quad b > 0, \]

where \( a \) and \( b \) are the scale and shape parameters.

The DataMining package class NaiveBayesClassifier uses GammaDistribution as a method to train continuous data.
**Properties**

**ScaleParameter**

```csharp
virtual public double ScaleParameter {get; }
```

**Description**

The maximum-likelihood estimate found for the gamma scale parameter.

**Property Value**

A `double` representing the maximum-likelihood estimate found for the gamma scale parameter.

---

**ShapeParameter**

```csharp
virtual public double ShapeParameter {get; }
```

**Description**

The maximum-likelihood estimate found for the gamma shape parameter.

**Property Value**

A `double` representing the maximum-likelihood estimate found for the gamma shape parameter.

---

**Constructor**

**GammaDistribution**

```csharp
public GammaDistribution()
```

**Description**


---

**Methods**

**Eval**

```csharp
virtual public double[] Eval(double[] xData)
```

**Description**

Fits a gamma probability distribution to `xData` and returns the probability density at each value.
Parameter

xData – A double array representing the points at which the gamma probability distribution function is to be evaluated.

Returns

A double array representing the gamma probability density at each value of xData.

Eval

virtual public double[] Eval(double[] xData, object[] parameters)

Description

Evaluates a gamma probability distribution with a given set of parameters at each point in xData and returns the probability density at each value.

Parameters

xData – A double array representing the points at which the gamma probability distribution function is to be evaluated.

parameters – An Object array representing the parameters used to evaluate the gamma distribution, see method GetParameters.

Returns

A double array representing the gamma probability density at each value of xData.

Eval

virtual public double Eval(double xData, object[] parameters)

Description

Evaluates a gamma probability density at a given point xData.

Parameters

xData – A double representing the point at which the gamma probability distribution function is to be evaluated.

parameters – An Object array representing the parameters used to evaluate the gamma distribution, see method GetParameters.

Returns

A double representing the gamma probability density at xData.

GetParameters

virtual public object[] GetParameters()

Description

Returns the current parameters of the gamma probability density function.
Returns
An Object array containing the parameters resulting from the last invocation of the (IDistribution) Eval method with the following signature, double[] Eval(double[] xData). This Object array can be used as input to the Eval methods that require an Object array of distribution parameters as input.

LogNormalDistribution Class


Evaluates a lognormal probability density for a given set of data.

LogNormalDistribution evaluates the lognormal probability density of a given set of data, xData. If parameters are not supplied, the Eval method fits the lognormal probability density function to the data by first calculating the mean and standard deviation. The lognormal probability density function is defined as:

\[ f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln(x) - \mu)^2}{2\sigma^2}} \]

where \( \mu \) and \( \sigma \) are the mean and standard deviation.

The DataMining package class NaiveBayesClassifier uses LognormalDistribution as a method to train continuous data.

See Also

Naive Bayes Example 1

Properties

Mean
virtual public double Mean {get; }

Description
The lognormal probability distribution mean parameter.

Property Value
A double representing the mean parameter.
**StandardDeviation**

virtual public double StandardDeviation {get; }

**Description**
The lognormal probability distribution standard deviation parameter.

**Property Value**
A double representing the standard deviation parameter.

**Constructor**

**LogNormalDistribution**

public LogNormalDistribution()

**Description**

**Methods**

**Eval**

virtual public double[] Eval(double[] xData)

**Description**
Fits a lognormal probability distribution to xData and returns the probability density at each value.

**Parameter**

xData – A double array representing the points at which the lognormal probability distribution function is to be evaluated.

**Returns**
A double array representing the lognormal probability density at each value of xData.

**Eval**

virtual public double[] Eval(double[] xData, object[] parameters)

**Description**
Evaluates a lognormal probability distribution with a given set of parameters at each point in xData and returns the probability density at each value.

**Parameters**

xData – A double array representing the points at which the lognormal probability distribution function is to be evaluated.

parameters – An Object array representing the parameters used to evaluate the lognormal distribution, see method GetParameters.
Returns
A double array representing the lognormal probability density at each value of xData.

Eval
virtual public double Eval(double xData, object[] parameters)

Description
Evaluates a lognormal probability density function at a given point xData.

Parameters
- xData – A double representing the point at which the lognormal probability distribution function is to be evaluated.
- parameters – An Object array representing the parameters used to evaluate the lognormal distribution, see method GetParameters.

Returns
A double representing the lognormal probability density at xData.

GetParameters
virtual public object[] GetParameters()

Description
Returns the current parameters of the lognormal probability density function.

Returns
An Object array containing the parameters resulting from the last invocation of the (IDistribution) Eval method with the following signature, double[] Eval(double[] xData). This Object array can be used as input to the Eval methods that require an Object array of distribution parameters as input.

PoissonDistribution Class

public class Imsl.Stat.PoissonDistribution :

Evaluates a Poisson probability density of a given set of data.

PoissonDistribution evaluates the Poisson probability density of a given set of data, xData. If parameters are not supplied, the Eval method fits the Poisson probability density function by first calculating theta, \( \theta \). The Poisson probability density function is defined as:

\[
f(x) = \frac{\theta^x e^{-\theta}}{x!}, x \geq 0 \text{ and } \theta > 0.
\]
The DataMining package class NaiveBayesClassifier uses PoissonDistribution as a method to train continuous data.

See Also

Naive Bayes Example 1

Property

Theta

```csharp
virtual public double Theta {get; }
```

Description
The mean number of successes in a given time period of the Poisson probability distribution.

Property Value
A double representing the mean number of successes in a given time period of the Poisson probability distribution.

Constructor

```csharp
public PoissonDistribution()
```

Description

Methods

Eval

```csharp
virtual public double[] Eval(double[] xData)
```

Description
Fits a Poisson probability distribution to xData and returns the probability density at each value.

Parameter

xData – A double array representing the points at which the Poisson probability distribution function is to be evaluated.
Returns
A double array representing the Poisson probability density at each value of xData.

**Eval**

`virtual public double[] Eval(double[] xData, object[] parameters)`

**Description**
Evaluates a Poisson probability distribution with a given set of parameters at each point in xData and returns the probability density at each value.

**Parameters**
- `xData` – A double array representing the points at which the Poisson probability distribution function is to be evaluated.
- `parameters` – An `Object` array representing the parameters used to evaluate the Poisson distribution, see method `GetParameters`.

**Returns**
A double array representing the Poisson probability density at each value of xData.

**Eval**

`virtual public double Eval(double xData, object[] parameters)`

**Description**
Evaluates a Poisson probability density function at a given point xData.

**Parameters**
- `xData` – A double representing the point at which the Poisson probability distribution function is to be evaluated.
- `parameters` – An `Object` array representing the parameters used to evaluate the Poisson distribution, see method `GetParameters`.

**Returns**
A double representing the Poisson probability density at xData.

**GetParameters**

`virtual public object[] GetParameters()`

**Description**
Returns the current parameters of the Poisson probability density function.

**Returns**
An `Object` array containing the parameters resulting from the last invocation of the (IDistribution) `Eval` method with the following signature, `double[] Eval(double[] xData)`. This `Object` array can be used as input to the `Eval` methods that require an `Object` array of distribution parameters as input.
Chapter 22: Random Number Generation

Types

class Random ................................................................. 1182
class Random.BaseGenerator ........................................... 1206
class MersenneTwister ....................................................... 1207
class MersenneTwister64 ..................................................... 1211
class FaureSequence .......................................................... 1216
interface IRandomSequence ............................................... 1221

Usage Notes

Overview of Random Number Generation

This chapter describes functions for the generation of random numbers that are useful for applications in simulation studies.

In the following discussions, the phrases random numbers, random deviates, deviates, and variates are used interchangeably. The phrase pseudorandom is sometimes used to emphasize that the numbers generated are really not random since they result from a deterministic process. The usefulness of pseudorandom numbers is derived from the similarity, in a statistical sense, of samples of the pseudorandom numbers to samples of observations from the specified distributions. In short, while the pseudorandom numbers are completely deterministic and repeatable, they simulate the realizations of independent and identically distributed random variables.

Class Imsl.Stat.Random extends System.Random. It adds the ability to generate random numbers with a number of different non-uniform distributions. The non-uniform random number generators use a uniform random number generator. The uniform random number generator in System.Random can be used. Imsl.Stat.Random adds a linear congruential generator. It is also possible to use another uniform generator as long as it implements the Random.BaseGenerator interface.

The MersenneTwister and MersenneTwister64 uniform random number generators generate random number series with very long periods. They both implement the Random.BaseGenerator interface and
so can be used as the base uniform generator in Imsl.Stat.Random.

The class FaureSequence generates the low-discrepancy Faure sequence. This is also called a quasi-random generator. The sequence is a series of points in n-dimensions, which are close to being as equally spaced as possible. Low-discrepancy refers to the difference from actually being as equally spaced as possible.

The FaureSequence implements the IRandomSequence interface. This interface defines a sequence of points in n-dimensions. It is used by the class Imsl.Math.HyperRectangleQuadrature to evaluate n-dimensional integrals.

Random Class

public class Imsl.Stat.Random : Random

Generate uniform and non-uniform random number distributions.

The non-uniform distributions are generated from a uniform distribution. By default, this class uses the uniform distribution generated by the base class System.Random. If the multiplier is set in this class then a multiplicative congruential method is used. The form of the generator is

\[ x_i \equiv c x_{i-1} \mod (2^{31} - 1) \]

Each \( x_i \) is then scaled into the unit interval (0,1). If the multiplier, \( c \), is a primitive root modulo \( 2^{31} - 1 \) (which is a prime), then the generator will have a maximal period of \( 2^{31} - 2 \). There are several other considerations, however. See Knuth (1981) for a good general discussion. Possible values for \( c \) are 16807, 397204094, and 950706376. The selection is made by the property Multiplier (p. 1184). Evidence suggests that the performance of 950706376 is best among these three choices (Fishman and Moore 1982).

Alternatively, one can select a 32-bit or 64-bit Mersenne Twister generator by first instantiating Imsl.Stat.MersenneTwister (p. 1207) or Imsl.Stat.MersenneTwister64 (p. 1211). These generators have a period of \( 2^{19937} - 1 \) and a 623-dimensional equidistribution property. See Matsumoto et al. 1998 for details.

The generation of uniform (0,1) numbers is done by the method NextFloat (p. 1191).

Nonuniform random numbers are generated using a variety of transformation procedures. All of the transformations used are exact (mathematically). The most straightforward transformation is the inverse CDF technique, but it is often less efficient than others involving acceptance/rejection and mixtures. See Kennedy and Gentle (1980) for discussion of these and other techniques.

Many of the nonuniform generators use different algorithms depending on the values of the parameters of the distributions. This is particularly true of the generators for discrete distributions. Schmeiser (1983) gives an overview of techniques for generating deviates from discrete distributions.

Extensive empirical tests of some of the uniform random number generators available in the Random class are reported by Fishman and Moore (1982 and 1986). Results of tests on the generator using the
multiplier 16807 are reported by Learmonth and Lewis (1973). If the user wishes to perform additional tests, the routines in Chapter 17, *Tests of Goodness of Fit*, may be of use. Often in Monte Carlo applications, it is appropriate to construct an ad hoc test that is sensitive to departures that are important in the given application. For example, in using Monte Carlo methods to evaluate a one-dimensional integral, autocorrelations of order one may not be harmful, but they may be disastrous in evaluating a two-dimensional integral. Although generally the routines in this chapter for generating random deviates from nonuniform distributions use exact methods, and, hence, their quality depends almost solely on the quality of the underlying uniform generator, it is often advisable to employ an ad hoc test of goodness of fit for the transformations that are to be applied to the deviates from the nonuniform generator.

Three methods are associated with copulas. A *copula* is a multivariate cumulative probability distribution (CDF) whose arguments are random variables uniformly distributed on the interval [0,1] corresponding to the probabilities (variates) associated with arbitrarily distributed marginal deviates. The copula structure allows the multivariate CDF to be partitioned into the copula, which has associated with it information characterizing the dependence among the marginal variables, and the set of separate marginal deviates, each of which has its own distribution structure.

Two methods, `NextGaussianCopula` and `NextStudentsTCopula`, allow the user to specify a correlation structure (in the form of a Cholesky matrix) which can be used to imprint correlation information on a sequence of multivariate random vectors. Each call to one of these methods returns a random vector whose elements (variates) are each uniformly distributed on the interval [0,1] and correlated according to a user-specified Cholesky matrix. These variate vector sequences may then be inverted to marginal deviate sequences whose distributions and imprinted correlations are user-specified. Method `NextGaussianCopula` generates a random Gaussian copula sequence by inverting uniform [0,1] random numbers to N(0,1) deviates vectors, imprinting each vector with the correlation information by multiplying it with the Cholesky matrix, and then using the N(0,1) CDF to map the imprinted deviates back to uniform [0,1] variates. Method `NextStudentsTCopula` inverts a vector of uniform [0,1] random numbers to a Student’s t deviate vector with mean 0 and user specified degrees of freedom `df` which is then imprinted with the Cholesky matrix and mapped back to uniform [0,1] variates.

The third copula method, `CanonicalCorrelation`, extracts a correlation matrix from a sequence of multivariate deviate vectors whose component marginals are arbitrarily distributed. This is accomplished by first extracting the empirical CDF from each of the marginal deviates and then using this CDF to map the deviates to uniform [0,1] variates which are then inverted to Normal (0,1) deviates. Each element $C_{ij}$ of the correlation matrix can then be extracted by averaging the products $z_i z_j$ of deviates $i$ and $j$ over the $t$-indexed sequence. The utility of method `CanonicalCorrelation` is that because the correlation matrix is derived from N(0,1) deviates, the correlation is unbiased, i.e. undistorted by the arbitrary marginal distribution structures of the original deviate vector sequences. This is important in such financial applications as portfolio optimization, where correlation is used to estimate and minimize risk.

The use of these copula methods is illustrated with RandomEx2.cs, which first uses method `NextGaussianCopula` to create a correlation imprinted sequence of random deviate vectors and then uses method `CanonicalCorrelation` to extract the correlation matrix from the imprinted sequence of vectors.
Properties

**Multiplier**

```csharp
public long Multiplier {get; set; }
```

**Description**
The multiplier for a linear congruential random number generator.

**Property Value**
A `long` which represents the random number generator multiplier.

**Remarks**
If not set, the multiplier has the value zero. If a multiplier is set then the linear congruential generator, defined in the base class `System.Random`, is replaced by the generator

```
seed = (multiplier*seed) mod (2^31 − 1)
```

See Donald Knuth, The Art of Computer Programming, Volume 2, for guidelines in choosing a multiplier. Some possible values are 16807, 397204094, 950706376.

**NumberOfProcessors**

```csharp
public int NumberOfProcessors {get; set; }
```

**Description**
Perform the parallel calculations with the maximum possible number of processors set to `NumberOfProcessors`.

**Property Value**
An `int` indicating the maximum possible number of processors to use.

**Remarks**
By default, `NumberOfProcessors = Environment.ProcessorCount`. If `NumberOfProcessors` is set to a number less than 1 or greater than `Environment.ProcessorCount`, `Environment.ProcessorCount` will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

Constructors

**Random**

```csharp
public Random()
```

**Description**
Constructor for the Random number generator class.

**Random**

```csharp
public Random(int seed)
```
Description
Constructor for the Random number generator class with supplied seed.

Parameter
seed – An int which represents the random number generator seed in the range of -2,147,483,647 to +2,147,483,647.

Random

Description
Constructor for the Random number generator class with an alternate basic number generator.

Parameter
baseGenerator – A BaseGenerator used to override the method Next.

Methods

CanonicalCorrelation
public double[][] CanonicalCorrelation(double[][] deviate)

Description
Method CanonicalCorrelation generates a canonical correlation matrix from an arbitrarily distributed multivariate deviate sequence with \( nvar \) deviate variables, \( nseq \) steps in the sequence, and a Gaussian Copula dependence structure.

Parameter
deviate – A double \( nseq \) by \( nvar \) array of input deviate values.

Remarks
Method CanonicalCorrelation first maps each of the \( j=1..nvar \) input deviate sequences \( \text{deviate}[k=1..nseq][j] \) into a corresponding sequence of variates, say \( \text{variate}[k][j] \) (where variates are values of the empirical cumulative probability function, \( CDF(x) \), defined as the probability that random deviate variable \( X \leq x \)). The variate matrix \( \text{variate}[k][j] \) is then mapped into Normal(0,1) distributed deviates \( \tilde{z}_{kj} \) using the method \( \text{InvCdf.Normal(variate[k][j])} \) and then the standard covariance estimator

\[
C_{ij} = \frac{1}{n_{seq}} \sum_{k=1}^{n_{seq}} \tilde{z}_{ki} \tilde{z}_{kj}
\]

is used to calculate the canonical correlation matrix \( \text{correlation} = \text{CanonicalCorrelation(deviate)} \), where \( C_{ij} = \text{correlation}[i][j] \) and \( n_{seq} = nseq \).

If a multivariate distribution has Gaussian marginal distributions, then the standard “empirical” correlation matrix given above is “unbiased”, i.e. an accurate measure of dependence among the variables. But when the marginal distributions depart significantly from Gaussian, i.e. are skewed or flattened, then the empirical correlation may become biased. One way to remove such bias from
dependence measures is to map the non-Gaussian-distributed marginal deviates to Gaussian N(0,1) deviates (by mapping the non-Gaussian marginal deviates to empirically derived marginal CDF variate values, then inverting the variates to N(0,1) deviates as described above), and calculating the standard empirical correlation matrix from these N(0,1) deviates as in the equation above. The resulting “(Gaussian) canonical correlation” matrix thereby avoids the bias that would occur if the empirical correlation matrix were extracted from the non-Gaussian marginal distributions directly.

The canonical correlation matrix may be of value in such applications as Markowitz portfolio optimization, where an unbiased measure of dependence is required to evaluate portfolio risk, defined in terms of the portfolio variance which is in turn defined in terms of the correlation among the component portfolio instruments.

The utility of the canonical correlation derives from the observation that a “copula” multivariate distribution with uniformly-distributed deviates (corresponding to the CDF probabilities associated with the marginal deviates) may be mapped to arbitrarily distributed marginals, so that an unbiased dependence estimator derived from one set of marginals (N(0,1) distributed marginals) can be used to represent the dependence associated with arbitrarily-distributed marginals. The “Gaussian Copula” (whose variate arguments are derived from N(0,1) marginal deviates) is a particularly useful structure for representing multivariate dependence.

This is demonstrated in Example 2 where method Random.NextGaussianCopula(chol) (where chol is a Cholesky object derived from a user-specified covariance matrix) is used to imprint correlation information on otherwise arbitrarily distributed and independent random sequences. Method Random.CanonicalCorrelation is then used to extract an unbiased correlation matrix from these imprinted deviate sequences.

Next
override public int Next(int maxValue)

Description
Returns a nonnegative pseudorandom int.

Parameter
maxValue – An int which specifies the upper bound of the random number to be generated.
maxValue must be greater than or equal to zero.

Returns
An int greater than or equal to zero and less than maxValue.

Next
override public int Next(int minValue, int maxValue)

Description
Returns a nonnegative pseudorandom int in the specified range.

Parameters
minValue – An int which specifies the lower bound of the random number returned.
maxValue – An int which specifies the upper bound of the random number to be generated.
maxValue must be greater than or equal to zero.
Returns
An int greater than or equal to minValue and less than maxValue; that is, the range of return values includes minValue but not maxValue. If minValue equals maxValue, minValue is returned.

NextBeta
virtual public double NextBeta(double p, double q)

Description
Generate a pseudorandom number from a beta distribution.

Parameters
p – A double which specifies the first beta distribution parameter, \( p > 0 \).
q – A double which specifies the second beta distribution parameter, \( q > 0 \).

Returns
A double which specifies a pseudorandom number from a beta distribution.

Remarks
Method NextBeta generates pseudorandom numbers from a beta distribution with parameters \( p \) and \( q \), both of which must be positive. The probability density function is

\[
f(x) = \frac{\Gamma(p+q)}{\Gamma(p)\Gamma(q)} x^{p-1} (1-x)^{q-1} \quad \text{for} \quad 0 \leq x \leq 1
\]

where \( \Gamma(\cdot) \) is the gamma function.

The algorithm used depends on the values of \( p \) and \( q \). Except for the trivial cases of \( p = 1 \) or \( q = 1 \), in which the inverse CDF method is used, all of the methods use acceptance/rejection. If \( p \) and \( q \) are both less than 1, the method of Johnk (1964) is used; if either \( p \) or \( q \) is less than 1 and the other is greater than 1, the method of Atkinson (1979) is used; if both \( p \) and \( q \) are greater than 1, algorithm BB of Cheng (1978), which requires very little setup time, is used.

The value returned is less than 1.0 and greater than \( \varepsilon \), where \( \varepsilon \) is the smallest positive number such that \( 1.0 - \varepsilon \) is less than 1.0.

NextBinomial
virtual public int NextBinomial(int n, double p)

Description
Generate a pseudorandom number from a Binomial distribution.

Parameters
n – A int which specifies the number of Bernoulli trials.
p – A double which specifies the probability of success on each trial, \( 0 < p < 1 \).

Returns
A int which specifies the pseudorandom number from a Binomial distribution.
Remarks

NextBinomial generates pseudorandom numbers from a Binomial distribution with parameters $n$ and $p$. $n$ and $p$ must be positive, and $p$ must be less than 1. The probability function (with $n = n$ and $p = p$) is

$$f(x) = \binom{n}{x} p^x (1 - p)^{n-x}$$

for $x = 0, 1, 2, \ldots, n$.

The algorithm used depends on the values of $n$ and $p$. If $np < 10$ or if $p$ is less than a machine epsilon, the inverse CDF technique is used; otherwise, the BTPE algorithm of Kachitvichyanukul and Schmeiser (see Kachitvichyanukul 1982) is used. This is an acceptance/rejection method using a composition of four regions. (TPE equals Triangle, Parallelogram, Exponential, left and right.)

NextCauchy

virtual public double NextCauchy()

Description

Generates a pseudorandom number from a Cauchy distribution.

Returns

A double which specifies a pseudorandom number from a Cauchy distribution.

Remarks

The probability density function is

$$f(x) = \frac{1}{\pi(1+x^2)}$$

Use of the inverse CDF technique would yield a Cauchy deviate from a uniform (0, 1) deviate, $u$, as $\tan[\pi(u - .5)]$. Rather than evaluating a tangent directly, however, NextCauchy generates two uniform (-1, 1) deviates, $x_1$ and $x_2$. These values can be thought of as sine and cosine values. If

$$x_1^2 + x_2^2$$

is less than or equal to 1, then $x_1/x_2$ is delivered as the Cauchy deviate; otherwise, $x_1$ and $x_2$ are rejected and two new uniform (-1, 1) deviates are generated. This method is also equivalent to taking the ratio of two independent normal deviates.

Deviates from the Cauchy distribution with median $t$ and first quartile $t - s$, that is, with density

$$f(x) = \frac{s}{\pi [s^2 + (x-t)^2]}$$

can be obtained by scaling the output from NextCauchy. To do this, first scale the output from NextCauchy by $S$ and then add $T$ to the result.
NextChiSquared
virtual public double NextChiSquared(double df)

Description
Generates a pseudorandom number from a Chi-squared distribution.

Parameter
   df – A double which specifies the number of degrees of freedom. It must be positive.

Returns
A double which specifies a pseudorandom number from a Chi-squared distribution.

Remarks
NextChiSquared generates pseudorandom numbers from a chi-squared distribution with df degrees of freedom. If df is an even integer less than 17, the chi-squared deviate r is generated as

\[ r = -2 \ln \left( \prod_{i=1}^{n} u_i \right) \]

where \( n = df/2 \) and the \( u_i \) are independent random deviates from a uniform (0, 1) distribution. If df is an odd integer less than 17, the chi-squared deviate is generated in the same way, except the square of a normal deviate is added to the expression above. If df is greater than 16 or is not an integer, and if it is not too large to cause overflow in the gamma random number generator, the chi-squared deviate is generated as a special case of a gamma deviate, using NextGamma. If overflow would occur in NextGamma, the chi-squared deviate is generated in the manner described above, using the logarithm of the product of uniforms, but scaling the quantities to prevent underflow and overflow.

NextDouble
override public double NextDouble()

Description
Generates the next pseudorandom number.

Returns
A double which specifies the next pseudorandom value from this random number generator’s sequence.

Remarks
If the multiplier is set then the multiplicative congruential method is used. Otherwise, super.Next(bits) is used. Where bits is the number of random bits required.

NextExponential
virtual public double NextExponential()

Description
Generates a pseudorandom number from a standard exponential distribution.

Returns
A double which specifies a pseudorandom number from a standard exponential distribution.
Remarks
The probability density function is \( f(x) = e^{-x}; \) for \( x > 0. \)

NextExponential uses an antithetic inverse CDF technique; that is, a uniform random deviate \( U \) is generated and the inverse of the exponential cumulative distribution function is evaluated at \( 1.0 - U \) to yield the exponential deviate.

Deviates from the exponential distribution with mean \( \text{THETA} \) can be generated by using NextExponential and then multiplying the result by \( \text{THETA}. \)

NextExponentialMix
virtual public double NextExponentialMix(double theta1, double theta2, double p)

Description
Generate a pseudorandom number from a mixture of two exponential distributions.

Parameters
theta1 – A double which specifies the mean of the exponential distribution that has the larger mean.
theta2 – A double which specifies the mean of the exponential distribution that has the smaller mean. theta2 must be positive and less than or equal to theta1.
p – A double which specifies the mixing parameter. It must satisfy \( 0 \leq p \leq \text{theta1}/(\text{theta1} - \text{theta2}). \)

Returns
A double which specifies a pseudorandom number from a mixture of the two exponential distributions.

Remarks
The probability density function is

\[
 f(x) = \frac{p}{\theta} e^{-x/\theta_1} + \frac{1 - p}{\theta_2} e^{-x/\theta_2} \quad \text{for} \quad x > 0
\]

where \( p = p, \theta_1 = \text{theta1}, \) and \( \theta_2 = \text{theta2}. \)

In the case of a convex mixture, that is, the case \( 0 < p < 1, \) the mixing parameter \( p \) is interpretable as a probability; and NextExponentialMix with probability \( p \) generates an exponential deviate with mean \( \theta_1, \) and with probability \( 1 - p \) generates an exponential with mean \( \theta_2. \) When \( p \) is greater than 1, but less than \( \theta_1/\theta_1 - \theta_2), then either an exponential deviate with mean \( \theta_2 \) or the sum of two exponentials with means \( \theta_1 \) and \( \theta_2 \) is generated. The probabilities are \( q = p - (p - 1)\theta_1/\theta_2 \) and \( 1 - q, \) respectively, for the single exponential and the sum of the two exponentials.

NextExtremeValue
virtual public double NextExtremeValue(double mu, double beta)

Description
Generate a pseudorandom number from an extreme value distribution.
Parameters
mu – A double scalar value representing the location parameter.
beta – A double scalar value representing the scale parameter.

Returns
A double pseudorandom number from an extreme value distribution

NextF
virtual public double NextF(double dfn, double dfd)

Description
Generate a pseudorandom number from the F distribution.

Parameters
dfn – A double, the numerator degrees of freedom. It must be positive.
dfd – A double, the denominator degrees of freedom. It must be positive.

Returns
A double, a pseudorandom number from an F distribution

NextFloat
public float NextFloat()

Description
Generates the next pseudorandom number.

Returns
A float which specifies the next pseudorandom value from this random number generator’s sequence.

Remarks
If the multiplier is set then the multiplicative congruential method is used. Otherwise, super.Next(bits) is used. Where bits is the number of random bits required.

NextGamma
virtual public double NextGamma(double a)

Description
Generates a pseudorandom number from a standard gamma distribution.

Parameter
a – A double which specifies the shape parameter of the gamma distribution. It must be positive.

Returns
A double which specifies a pseudorandom number from a standard gamma distribution.
Remarks

Method NextGamma generates pseudorandom numbers from a gamma distribution with shape parameter \( a \). The probability density function is

\[
P = \frac{1}{\Gamma(a)} \int_0^x e^{-t}t^{a-1}dt
\]

Various computational algorithms are used depending on the value of the shape parameter \( a \). For the special case of \( a = 0.5 \), squared and halved normal deviates are used; and for the special case of \( a = 1.0 \), exponential deviates (from method NextExponential) are used. Otherwise, if \( a \) is less than 1.0, an acceptance-rejection method due to Ahrens, described in Ahrens and Dieter (1974), is used; if \( a \) is greater than 1.0, a ten-region rejection procedure developed by Schmeiser and Lal (1980) is used.

The Erlang distribution is a standard gamma distribution with the shape parameter having a value equal to a positive integer; hence, NextGamma generates pseudorandom deviates from an Erlang distribution with no modifications required.

NextGaussianCopula

virtual public double[] NextGaussianCopula(Imsl.Math.Cholesky chol)

Description

Generate pseudorandom numbers from a Gaussian Copula distribution.

Parameter

\chol – A Cholesky object containing the Cholesky factorization of the correlation matrix of order \( k \).

Returns

A double array which contains the pseudorandom numbers from a multivariate Gaussian Copula distribution.

Remarks

NextGaussianCopula generates pseudorandom numbers from a multivariate Gaussian Copula distribution which are uniformly distributed on the interval \((0,1)\) representing the probabilities associated with \( N(0,1) \) deviates imprinted with correlation information from input Cholesky object \( \text{chol} \). Cholesky matrix \( R \) is defined as the “square root” of a user-defined correlation matrix, that is \( R \) is an upper triangular matrix such that the transpose of \( R \) times \( R \) is the correlation matrix.

First, a length \( k \) vector of independent random normal deviates with mean 0 and variance 1 is generated, and then this deviate vector is post-multiplied by cholesky matrix \( R \). Finally, the Cholesky-imprinted random \( N(0,1) \) deviates are mapped to output probabilities using the \( N(0,1) \) cumulative distribution function (CDF).

Random deviates from arbitrary marginal distributions which are imprinted with the correlation information contained in Cholesky matrix \( R \) can then be generated by inverting the output probabilities using user-specified inverse CDF functions.

NextGeometric

virtual public int NextGeometric(double p)
**Description**

Generate a pseudorandom number from a geometric distribution.

**Parameter**

- \( p \) – A double which specifies the probability of success on each trial, \( 0 < p \leq 1 \).

**Returns**

A int which specifies a pseudorandom number from a geometric distribution.

**Remarks**

NextGeometric generates pseudorandom numbers from a geometric distribution with parameter \( p \), where \( P = p \) is the probability of getting a success on any trial. A geometric deviate can be interpreted as the number of trials until the first success (including the trial in which the first success is obtained). The probability function is

\[
   f(x) = P(1 - P)^{x-1}
\]

for \( x = 1, 2, \ldots \) and \( 0 < P < 1 \).

The geometric distribution as defined above has mean \( 1/P \).

The \( i \)-th geometric deviate is generated as the smallest integer not less than \( \log(U_i)/\log(1 - P) \), where the \( U_i \) are independent uniform (0, 1) random numbers (see Knuth, 1981).

The geometric distribution is often defined on 0, 1, 2, ..., with mean \( (1 - P)/P \). Such deviates can be obtained by subtracting 1 from each element returned value.

**NextHypergeometric**

virtual public int NextHypergeometric(int n, int m, int l)

**Description**

Generate a pseudorandom number from a hypergeometric distribution.

**Parameters**

- \( n \) – A int which specifies the number of items in the sample, \( n > 0 \).
- \( m \) – A int which specifies the number of special items in the population, or lot, \( m > 0 \).
- \( l \) – A int which specifies the number of items in the lot, \( l > \max(n,m) \).

**Returns**

A int which specifies the number of special items in a sample of size \( n \) drawn without replacement from a population of size \( l \) that contains \( m \) such special items.
Remarks
Method NextHypergeometric generates pseudorandom numbers from a hypergeometric distribution with parameters \(n, m, \) and \(l\). The hypergeometric random variable \(x\) can be thought of as the number of items of a given type in a random sample of size \(n\) that is drawn without replacement from a population of size \(l\) containing \(m\) items of this type. The probability function is

\[
f(x) = \frac{\binom{m}{x} \binom{l-m}{n-x}}{\binom{l}{n}}
\]

for \(x = \max(0, n-l+m), 1, 2, \ldots, \min(n,m)\).

If the hypergeometric probability function with parameters \(n, m, \) and \(l\) evaluated at \(n - l + m\) (or at 0 if this is negative) is greater than the machine epsilon, and less than 1.0 minus the machine epsilon, then NextHypergeometric uses the inverse CDF technique. The method recursively computes the hypergeometric probabilities, starting at \(x = \max(0, n-l+m)\) and using the ratio \(f(x = x + 1)/f(x = x)\) (see Fishman 1978, page 457).

If the hypergeometric probability function is too small or too close to 1.0, then NextHypergeometric generates integer deviates uniformly in the interval \([1, l - i]\), for \(i = 0, 1, \ldots\); and at the \(I\)-th step, if the generated deviate is less than or equal to the number of special items remaining in the lot, the occurrence of one special item is tallied and the number of remaining special items is decreased by one. This process continues until the sample size or the number of special items in the lot is reached, whichever comes first. This method can be much slower than the inverse CDF technique. The timing depends on \(n\).

If \(n\) is more than half of \(l\) (which in practical examples is rarely the case), the user may wish to modify the problem, replacing \(n\) by \(l - n\), and to consider the deviates to be the number of special items not included in the sample.

NextLogarithmic

virtual public int NextLogarithmic(double a)

Description
Generate a pseudorandom number from a logarithmic distribution.

Parameter

\(a\) – A double which specifies the parameter of the logarithmic distribution, \(0 < a < 1\).

Returns

A int which specifies a pseudorandom number from a logarithmic distribution.

Remarks
Method NextLogarithmic generates pseudorandom numbers from a logarithmic distribution with parameter \(a\). The probability function is

\[
f(x) = \frac{a^x}{x\ln(1-a)}
\]

for \(x = 1, 2, 3, \ldots, \) and \(0 < a < 1\).
The methods used are described by Kemp (1981) and depend on the value of \( a \). If \( a \) is less than 0.95, Kemp’s algorithm LS, which is a “chop-down” variant of an inverse CDF technique, is used. Otherwise, Kemp’s algorithm LK, which gives special treatment to the highly probable values of 1 and 2, is used.

### NextLogNormal

`virtual public double NextLogNormal(double mean, double stdev)`

**Description**

Generate a pseudorandom number from a lognormal distribution.

**Parameters**

- `mean` – A `double` which specifies the mean of the underlying normal distribution.
- `stdev` – A `double` which specifies the standard deviation of the underlying normal distribution. It must be positive.

**Returns**

A `double` which specifies a pseudorandom number from a lognormal distribution.

**Remarks**

Method `NextLogNormal` generates pseudorandom numbers from a lognormal distribution with parameters `mean` and `stdev`. The scale parameter in the underlying normal distribution, `stdev`, must be positive. The method is to generate normal deviates with mean `mean` and standard deviation `stdev` and then to exponentiate the normal deviates.

With \( \mu = \text{mean} \) and \( \sigma = \text{stdev} \), the probability density function for the lognormal distribution is

\[
f(x) = \frac{1}{\sigma x \sqrt{2\pi}} \exp \left[ -\frac{1}{2\sigma^2} (\ln x - \mu)^2 \right] \quad \text{for } x > 0
\]

The mean and variance of the lognormal distribution are \( \exp(\mu + \sigma^2/2) \) and \( \exp(2\mu + 2\sigma^2) - \exp(2\mu + \sigma^2) \), respectively.

### NextMultivariateNormal

`virtual public double[] NextMultivariateNormal(Imsl.Math.Cholesky matrix)`

**Description**

Generate pseudorandom numbers from a multivariate normal distribution.

**Parameter**

- `matrix` – The Cholesky factorization of the variance-covariance matrix of order \( k \).

**Returns**

A `double` array which contains the pseudorandom numbers from a multivariate normal distribution.

---

**Random Number Generation**

---

Random • 1195
Remarks

NextMultivariateNormal generates pseudorandom numbers from a multivariate normal distribution with mean vector consisting of all zeroes and variance-covariance matrix whose Cholesky factor (or “square root”) is matrix; that is, matrix is an upper triangular matrix such that the transpose of matrix times matrix is the variance-covariance matrix. First, independent random normal deviates with mean 0 and variance 1 are generated, and then the matrix containing these deviates is post-multiplied by matrix.

Deviates from a multivariate normal distribution with means other than zero can be generated by using NextMultivariateNormal and then by adding the means to the deviates.

NextNegativeBinomial

virtual public int NextNegativeBinomial(double rk, double p)

Description

Generate a pseudorandom number from a negative Binomial distribution.

Parameters

rk – A double which specifies the negative binomial parameter, rk > 0.
p – A double which specifies the probability of success on each trial. It must be greater than machine precision and less than one.

Returns

A int which specifies the pseudorandom number from a negative binomial distribution. If rk is an integer, the deviate can be thought of as the number of failures in a sequence of Bernoulli trials before rk successes occur.

Remarks

Method NextNegativeBinomial generates pseudorandom numbers from a negative Binomial distribution with parameters rk and p. rk and p must be positive and p must be less than 1. The probability function with \((r = rk)\) and \((p = p)\) is

\[
f(x) = \binom{r+x-1}{x} (1-p)^r p^x
\]

for \(x = 0, 1, 2, \ldots\).

If \(r\) is an integer, the distribution is often called the Pascal distribution and can be thought of as modeling the length of a sequence of Bernoulli trials until \(r\) successes are obtained, where \(p\) is the probability of getting a success on any trial. In this form, the random variable takes values \(r, r + 1, r + 2, \ldots\) and can be obtained from the negative binomial random variable defined above by adding \(r\) to the negative binomial variable. This latter form is also equivalent to the sum of \(r\) geometric random variables defined as taking values 1, 2, 3, \ldots.

If \(rp/(1-p)\) is less than 100 and \((1 - p)^r\) is greater than the machine epsilon, NextNegativeBinomial uses the inverse CDF technique; otherwise, for each negative binomial deviate, NextNegativeBinomial generates a gamma \((r, p/(1 - p))\) deviate \(y\) and then generates a Poisson deviate with parameter \(y\).
**NextNormal**

virtual public double NextNormal()  

**Description**  
Generate a pseudorandom number from a standard normal distribution using an inverse CDF method.  

**Returns**  
A double which represents a pseudorandom number from a standard normal distribution.  

**Remarks**  
In this method, a uniform (0,1) random deviate is generated, then the inverse of the normal distribution function is evaluated at that point using InverseNormal. This method is slower than the acceptance/rejection technique used in NextNormalAR to generate standard normal deviates. Deviates from the normal distribution with mean $x_m$ and standard deviation $x_{std}$ can be obtained by scaling the output from NextNormal. To do this first scale the output of NextNormal by $x_{std}$ and then add $x_m$ to the result.

**NextNormalAR**

virtual public double NextNormalAR()  

**Description**  
Generate a pseudorandom number from a standard normal distribution using an acceptance/rejection method.  

**Returns**  
A double which represents a pseudorandom number from a standard normal distribution.  

**Remarks**  
NextNormalAR generates pseudorandom numbers from a standard normal (Gaussian) distribution using an acceptance/rejection technique due to Kinderman and Ramage (1976). In this method, the normal density is represented as a mixture of densities over which a variety of acceptance/rejection methods due to Marsaglia (1964), Marsaglia and Bray (1964), and Marsaglia, MacLaren, and Bray (1964) are applied. This method is faster than the inverse CDF technique used in NextNormal to generate standard normal deviates. Deviates from the normal distribution with mean $x_m$ and standard deviation $x_{std}$ can be obtained by scaling the output from NextNormalAR. To do this first scale the output of NextNormalAR by $x_{std}$ and then add $x_m$ to the result.

**NextPoisson**

virtual public int NextPoisson(double theta)  

**Description**  
Generate a pseudorandom number from a Poisson distribution.  

**Parameter**  
theta – A double which specifies the mean of the Poisson distribution, $\theta > 0$.  

**Returns**  
A int which specifies a pseudorandom number from a Poisson distribution.
Remarks
Method NextPoisson generates pseudorandom numbers from a Poisson distribution with parameter theta. theta, which is the mean of the Poisson random variable, must be positive. The probability function (with $\theta = \theta$) is

$$f(x) = e^{-\theta} \frac{\theta^x}{x!}$$

for $x = 0, 1, 2, \ldots$

If theta is less than 15, NextPoisson uses an inverse CDF method; otherwise the PTPE method of Schmeiser and Kachitvichyanukul (1981) (see also Schmeiser 1983) is used.

The PTPE method uses a composition of four regions, a triangle, a parallelogram, and two negative exponentials. In each region except the triangle, acceptance/rejection is used. The execution time of the method is essentially insensitive to the mean of the Poisson.

**NextRayleigh**

```csharp
virtual public double NextRayleigh(double alpha)
```

**Description**
Generate a pseudorandom number from a Rayleigh distribution.

**Parameter**
- alpha – A double which specifies the scale parameter of the Rayleigh distribution

**Returns**
A double, a pseudorandom number from a Rayleigh distribution

**Remarks**
Method NextRayleigh generates pseudorandom numbers from a Rayleigh distribution with scale parameter alpha.

**NextStudentsT**

```csharp
virtual public double NextStudentsT(double df)
```

**Description**
Generate a pseudorandom number from a Student’s t distribution.

**Parameter**
- df – A double which specifies the number of degrees of freedom. It must be positive.

**Returns**
A double which specifies a pseudorandom number from a Student’s t distribution.
Remarks
NextStudentsT generates pseudo-random numbers from a Student’s t distribution with df degrees of freedom, using a method suggested by Kinderman, Monahan, and Ramage (1977). The method (“TMX” in the reference) involves a representation of the t density as the sum of a triangular density over (-2, 2) and the difference of this and the t density. The mixing probabilities depend on the degrees of freedom of the t distribution. If the triangular density is chosen, the variate is generated as the sum of two uniforms; otherwise, an acceptance/rejection method is used to generate a variate from the difference density.

For degrees of freedom less than 100, NextStudentsT requires approximately twice the execution time as NextNormalAR, which generates pseudorandom normal deviates. The execution time of NextStudentsT increases very slowly as the degrees of freedom increase. Since for very large degrees of freedom the normal distribution and the t distribution are very similar, the user may find that the difference in the normal and the t does not warrant the additional generation time required to use NextStudentsT instead of NextNormalAR.

NextStudentsTCopula
virtual public double[] NextStudentsTCopula(double df, Imsl.Math.Cholesky chol)

Description
Generate pseudorandom numbers from a Student’s t Copula distribution.

Parameters
- df – A double which specifies the degrees of freedom parameter.
- chol – A Cholesky object containing the Cholesky factorization of the correlation matrix of order k.

Returns
A double array which contains the pseudorandom numbers from a multivariate Student’s t Copula distribution with df degrees of freedom.

Remarks
NextStudentsTCopula generates pseudorandom numbers from a multivariate Student’s t Copula distribution which are uniformly distributed on the interval (0,1) representing the probabilities associated with deviates from Student’s t distributions with df degrees of freedom imprinted with correlation information from the input Cholesky object chol. Cholesky matrix R is defined as the “square root” of a user-defined correlation matrix, that is R is an upper triangular matrix such that the transpose of R times R is the correlation matrix. First, a length k vector of independent random Student’s t deviates with mean 0 and df degrees of freedom is generated, and then this deviate vector is post-multiplied by Cholesky matrix R. Finally, the Cholesky-imprinted random Student’s t deviates are mapped to output probabilities using the Student’s t cumulative distribution function (CDF) with df degrees of freedom.

Random deviates from arbitrary marginal distributions which are imprinted with the correlation information contained in Cholesky matrix R can then be generated by inverting the output probabilities using user-specified inverse CDF functions.

NextTriangular
virtual public double NextTriangular()
**Description**
Generate a pseudorandom number from a triangular distribution on the interval (0,1).

**Returns**
A double which specifies a pseudorandom number from a triangular distribution on the interval (0,1).

**Remarks**
The probability density function is \( f(x) = 4x \), for \( 0 \leq x \leq 0.5 \), and \( f(x) = 4(1-x) \), for \( 0.5 < x \leq 1 \).

**NextTriangular**
virtual public double NextTriangular()

**Description**
Generate a pseudorandom number from a triangular distribution.

**Parameter**
- \( c \) – A double which specifies the parameter of the von Mises distribution, \( p > 7.4e-9 \).

**Returns**
A double which specifies a pseudorandom number from a von Mises distribution.

**Remarks**
Method NextVonMises generates pseudorandom numbers from a von Mises distribution with parameter \( c \), which must be positive. With \( c = C \), the probability density function is

\[
 f(x) = \frac{1}{2\pi I_0(c)} \exp[c \cos(x)] \text{ for } -\pi < x < \pi
\]

where \( I_0(c) \) is the modified Bessel function of the first kind of order 0. The probability density equals 0 outside the interval \((-\pi, \pi)\).

The algorithm is an acceptance/rejection method using a wrapped Cauchy distribution as the majorizing distribution. It is due to Best and Fisher (1979).

**NextVonMises**
virtual public double NextVonMises(double c)

**Description**
Generate a pseudorandom number from a von Mises distribution.

**Parameter**
- \( c \) – A double which specifies the parameter of the von Mises distribution, \( p > 7.4e-9 \).

**Returns**
A double which specifies a pseudorandom number from a von Mises distribution.

**Remarks**
Method NextVonMises generates pseudorandom numbers from a von Mises distribution with parameter \( c \), which must be positive. With \( c = C \), the probability density function is

\[
 f(x) = \frac{1}{2\pi I_0(c)} \exp[c \cos(x)] \text{ for } -\pi < x < \pi
\]

where \( I_0(c) \) is the modified Bessel function of the first kind of order 0. The probability density equals 0 outside the interval \((-\pi, \pi)\).

The algorithm is an acceptance/rejection method using a wrapped Cauchy distribution as the majorizing distribution. It is due to Best and Fisher (1979).

**NextWeibull**
virtual public double NextWeibull(double a)

**Description**
Generate a pseudorandom number from a Weibull distribution.

**Parameter**
- \( a \) – A double which specifies the shape parameter of the Weibull distribution, \( a > 0 \).

**Returns**
A double which specifies a pseudorandom number from a Weibull distribution.
Remarks
Method `NextWeibull` generates pseudorandom numbers from a Weibull distribution with shape parameter \( a \). The probability density function is

\[
f(x) = A x^{A-1} e^{-x^A} \text{ for } x \geq 0
\]

`NextWeibull` uses an antithetic inverse CDF technique to generate a Weibull variate; that is, a uniform random deviate \( U \) is generated and the inverse of the Weibull cumulative distribution function is evaluated at \( 1.0 - u \) to yield the Weibull deviate.

Deviates from the two-parameter Weibull distribution with shape parameter \( a \) can be generated by using `NextWeibull` and then multiplying the result by \( b \).

The Rayleigh distribution with probability density function,

\[
r(x) = \frac{1}{\alpha^2} x e^{-x^2/2\alpha^2} \text{ for } x \geq 0
\]

is the same as a Weibull distribution with shape parameter \( a \) equal to 2 and scale parameter \( b \) equal to \( \sqrt{2\alpha} \)

hence, `NextWeibull` and simple multiplication can be used to generate Rayleigh deviates.

**NextZigguratNormalAR**

```csharp
virtual public double NextZigguratNormalAR()
```

**Description**
Generates pseudorandom numbers using the Ziggurat method.

**Returns**
A `double` containing the random normal deviate.

**Remarks**
The `NextZigguratNormalAR` method cuts the density into many small pieces. For each random number generated, an interval is chosen at random and a random normal is generated from the chosen interval. In this implementation, the density is cut into 256 pieces, but symmetry is used so that only 128 pieces are needed by the computation. Following Doornik (2005), different uniform random deviates are used to determine which slice to use and to determine the normal deviate from the slice.

**Skip**

```csharp
virtual public void Skip(int n)
```

**Description**
Resets the seed to skip ahead in the base linear congruential generator.
**Parameter**

\( n \) – An int which specifies the number of random deviates to skip.

**Remarks**

This method can be used only if a linear congruential multiplier is explicitly defined by a call to Multiplier (p. 1184).

The method skips ahead in the deviates returned by the protected method Random.Next. The public methods use Next(int) as their source of uniform random deviates. Some methods call it more than once. For instance, each call to NextDouble (p. 1189) calls it twice.

**Example: Random Number Generation**

In this example, a discrete normal random sample of size 1000 is generated via NextNormal. After the ChiSquaredTest constructor is called, the random observations are added to the test one at a time to simulate streaming data. The Chi-squared test is performed using Cdf.Normal as the cumulative distribution function object to see how well the random numbers fit the normal distribution.

```csharp
using System;
using Imsl.Stat;

public class RandomEx1 : ICdfFunction
{
    public double CdfFunction(double x)
    {
        return Cdf.Normal(x);
    }

    public static void Main(String[] args)
    {
        int nObservations = 1000;
        ICdfFunction normal = new RandomEx1();
        ChiSquaredTest test = new ChiSquaredTest(normal, 10, 0);
        for (int k = 0; k < nObservations; k++)
        {
            test.Update(r.NextNormal(), 1.0);
        }

        double p = test.P;
        Console.Out.WriteLine("The P-value is " + p);
    }
}
```

**Output**

The P-value is 0.496307043723263
Example 2: Using Copulas to imprint and extract correlation information

This example uses method Random.NextGaussianCopula to generate a multivariate sequence GCdevt\[k=0..nseq-1][j=0..nvar-1]\ whose marginal distributions are user-defined and imprinted with a user-specified correlation matrix CorrMtrxIn[i=0..nvar-1][j=0..nvar-1] and then uses method Random.Covariance to extract from this multivariate random sequence a canonical correlation matrix CorrMtrx[i=0..nvar-1][j=0..nvar-1].

This example illustrates two useful copula related procedures. The first procedure generates a random multivariate sequence with arbitrary user-defined marginal deviates whose dependence is specified by a user-defined correlation matrix. The second procedure is the inverse of the first: an arbitrary multivariate deviate input sequence is first mapped to a corresponding sequence of empirically derived variates, i.e. cumulative distribution function values representing the probability that each random variable has a value less than or equal to the input deviate. The variates are then inverted, using the inverse Normal(0,1) function, to N(0,1) deviates; and finally, a canonical covariance matrix is extracted from the multivariate N(0,1) sequence using the standard sum of products.

This example demonstrates that the nextGaussianCopula method correctly imbeds the user-defined correlation information into an arbitrary marginal distribution sequence by extracting the canonical correlation from these sequences and showing that they differ from the original correlation matrix by a small relative error, which generally decreases as the number of multivariate sequence vectors increases.

```csharp
using System;
using Imsl.Math;
using Imsl.Stat;

public class RandomEx2
{
    internal static Imsl.Stat.Random IMSLRandom()
    {
        r.Multiplier = 16807;
        return r;
    }

    public static void Main(string[] args)
    {
        double[,] CorrMtrxIn = new double[,]{
            {1.0, -0.9486832980505138, 0.8164965809277261},
            {-0.9486832980505138, 1.0, -0.6454972243679028},
            {0.8164965809277261, -0.6454972243679028, 1.0}};

        int nvar = 3;

        Console.WriteLine("Random Example 2:");
        Console.WriteLine();

        for (int i = 0; i < nvar; i++)
        {
            for (int j = 0; j < i; j++)
            {
```
Console.WriteLine("CorrMtrxIn[" + i + "," + j + "] = " + CorrMtrxIn[i, j]);
}

PrintMatrixFormat pmf = new PrintMatrixFormat();
new PrintMatrix("Input Correlation Matrix: ").Print(pmf, CorrMtrxIn);
Console.WriteLine("Correlation Matrices calculated from");
Console.WriteLine(" Gaussian Copula imprinted multivariate sequence:");
Console.WriteLine();

// Compute the Cholesky factorization of CorrMtrxIn
Cholesky CholMtrx = new Cholesky(CorrMtrxIn);
for (int kmax = 500; kmax < 1000000; kmax *= 10)
{
    Console.WriteLine("# vectors in multivariate sequence: " + kmax);
    double[][] GCvart = new double[kmax][];
    double[][] GCdevt = new double[kmax][];
    for (int i2 = 0; i2 < kmax; i2++)
    {
        GCdevt[i2] = new double[nvar];
    }
    IMsl.Stat.Random r = IMSLRandom();
    for (int k = 0; k < kmax; k++)
    {
        GCvart[k] = r.NextGaussianCopula(CholMtrx); //probs
        for (int j = 0; j < nvar; j++)
        {
            /*
            * invert Gaussian Copula probabilities to deviates using
            * variable-specific inversions: j = 0: Chi Square; 1: F;
            * 2: Normal(0,1); will end up with deviate sequences ready
            * for mapping to canonical correlation matrix:
            */
            if (j == 0)
            {
                //convert probs into ChiSquare(df=10) deviates:
                GCdevt[k][j] = InvCdf.Chi(GCvart[k][j], 10.0);
            }
            else if (j == 1)
            {
                //convert probs into F(dfn=15,dfd=10) deviates:
                GCdevt[k][j] = InvCdf.F(GCvart[k][j], 15.0, 10.0);
            }
            else
            {
                //convert probs into Normal(mean=0,variance=1) deviates:
                GCdevt[k][j] = InvCdf.Normal(GCvart[k][j]);
            }
        }
    }
}
* extract Canonical Correlation matrix from arbitrarily distributed
* deviate sequences GCdevt[k=0..kmax-1][j=0..nvar-1] which have been
* imprinted with CorrMtrxIn[i=1..nvar][j=1..nvar] above:
*/
double[][] CorrMtrx = r.CanonicalCorrelation(GCdevt);
double relerr;
for (int i = 0; i < nvar; i++)
{
    for (int j = 0; j < i; j++)
    {
        relerr = Math.Abs(1.0 -
            (CorrMtrx[i][j] / CorrMtrxIn[i, j]));
        Console.WriteLine("CorrMtrx[" + i + "][" + j + "] = " +
            CorrMtrx[i][j] + "; relerr = " + relerr);
    }
    new PrintMatrix("Correlation Matrix: ").Print(pmf, CorrMtrx);
}

Output
Random Example 2:
CorrMtrxIn[1,0] = -0.948683298050514
CorrMtrxIn[2,0] = 0.816496580927726
CorrMtrxIn[2,1] = -0.645497224367903
Input Correlation Matrix:
  0  1  2
0 1.000000000 -0.948683298 0.816496581
1 -0.948683298 1.000000000 -0.645497224
2 0.816496581 -0.645497224 1.000000000

Correlation Matrices calculated from
Gaussian Copula imprinted multivariate sequence:
# vectors in multivariate sequence: 500
CorrMtrx[1][0] = -0.9502956568146; relerr = 0.00169957417218125
CorrMtrx[2][0] = 0.805260514673251; relerr = 0.0137613145197848
CorrMtrx[2][1] = -0.640202740166643; relerr = 0.00820217965529457
Correlation Matrix:
  0  1  2
0 1.000000000 -0.950295656 0.805260515
1 -0.950295656 1.000000000 -0.640202740
2 0.805260515 -0.640202740 1.000000000

# vectors in multivariate sequence: 5000
CorrMtrx[1][0] = -0.948611734634931; relerr = 7.54347480657058E-05
CorrMtrx[2][0] = 0.815532446740145; relerr = 0.00118081840157325
CorrMtrx[2][1] = -0.646255361981671; relerr = 0.00117450174090306
Correlation Matrix:
  0  1  2
0 1.000000000 -0.948611734 0.815532447
1 -0.948611734 1.000000000 -0.646255362

Random Number Generation • 1205
2 0.815532447 -0.646255362 1.000000000

# vectors in multivariate sequence: 50000
CorrMtrx[1][0] = -0.948314953115713; relerr = 0.00038826965285188
CorrMtrx[2][0] = 0.817827046711005; relerr = 0.00162948114463246
CorrMtrx[2][1] = -0.646669093465958; relerr = 0.00181545180028175

Correlation Matrix:
0 1 2
0 1.000000000 -0.948314953 0.817827047
1 -0.948314953 1.000000000 -0.646669093
2 0.817827047 -0.646669093 1.000000000

# vectors in multivariate sequence: 500000
CorrMtrx[1][0] = -0.94873295551488; relerr = 5.23435634081082E-05
CorrMtrx[2][0] = 0.81728795761655; relerr = 0.000969234540976194
CorrMtrx[2][1] = -0.646929884520875; relerr = 0.00221946756529401

Correlation Matrix:
0 1 2
0 1.000000000 -0.948732956 0.817287958
1 -0.948732956 1.000000000 -0.646929885
2 0.817287958 -0.646929885 1.000000000

---

**Random.BaseGenerator Interface**


Base pseudorandom number.

**Methods**

**Next**

abstract public int Next()

**Description**
Generates the next pseudorandom number.

**Returns**
The next pseudorandom value from this random number generator’s sequence.

**NextDouble**

abstract public double NextDouble()

**Description**
Generates the next pseudorandom double.
Returns
A double which is the next pseudorandom value from this random number generator’s sequence.

NextFloat
abstract public float NextFloat()

Description
Generates the next pseudorandom float.

Returns
A float which is the next pseudorandom value from this random number generator’s sequence.

MersenneTwister Class


A 32-bit Mersenne Twister generator.

By default, the class Imsl.Stat.Random (p. 1182) uses the uniform distribution generated by the base class System.Random. Alternatively, one can instantiate Imsl.Stat.MersenneTwister (p. 1207) or Imsl.Stat.MersenneTwister64 (p. 1211) to generate uniform pseudorandom numbers via the Mersenne Twister algorithm. These generators have a period of $2^{19937} - 1$ and a 623-dimensional equidistribution property. See Matsumoto et al. 1998 for details. The series of random numbers can be generated using a seed for initialization or by using an array of type int. This generator can be used to generate non-uniform distributions by creating an Imsl.Stat.Random (p. 1182) object using an instance of this class as an argument to the constructor. One can also save the state of the generator at initialization to be re-used later.

This C# code was translated from the following C program.

A C-program for MT19937, with initialization improved 2002/1/26. Coded by Takuji Nishimura and Makoto Matsumoto.

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Any feedback is very welcome.

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
mailto: m-mat@math.sci.hiroshima-u.ac.jp

Constructors

**MersenneTwister**

*public MersenneTwister(int s)*

**Description**

 Initializes the 32-bit Mersenne Twister generator using a seed.

**Parameter**

 *s* – An `int` which contains the seed that is used to initialize the 32-bit Mersenne Twister generator.

**MersenneTwister**

*public MersenneTwister(uint s)*

**Description**

 Initializes the 32-bit Mersenne Twister generator using a seed.

**Parameter**

 *s* – A `uint` which contains the seed that is used to initialize the 32-bit Mersenne Twister generator.

**MersenneTwister**

*public MersenneTwister(int[] key)*

**Description**

 Initializes the 32-bit Mersenne Twister generator using an array.
Parameter
   key – An int array used to initialize the 32-bit Mersenne Twister generator.

MersenneTwister
public MersenneTwister(uint[] key)

Description
Initializes the 32-bit Mersenne Twister generator using an array.

Parameter
   key – A uint array used to initialize the 32-bit Mersenne Twister generator.

Methods

Clone
Final public object Clone()

Description
Returns a clone of this object.

Returns
An Object which is a clone of this MersenneTwister object.

Next
virtual public int Next()

Description
Returns a nonnegative pseudorandom int.

Returns
An int greater than or equal to zero and less than System.Int32.MaxValue.

NextDouble
virtual public double NextDouble()

Description
Returns a random number between 0.0 and 1.0.

Returns
A double greater than or equal to 0.0, and less than 1.0.

Remarks
Only the first 32 bits of the double are pseudorandom.

NextFloat
virtual public float NextFloat()
Description
Returns a random number between 0.0 and 1.0.

Returns
A float greater than or equal to 0.0, and less than 1.0.

Example: Mersenne Twister Random Number Generation
In this example, four simulation streams are generated. The first series is generated with the seed used for initialization. The second series is generated using an array for initialization. The third series is obtained by resetting the generator back to the state it had at the beginning of the second stream. Therefore, the second and third streams are identical. The fourth stream is obtained by resetting the generator back to its original, uninitialized state, and having it reinitialize using the seed. The first and fourth streams are therefore the same.

```csharp
using System;
using System.IO;
using System.Runtime.Serialization;
using Imsl.Stat;

public class MersenneTwisterEx1
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    static void Main(string[] args)
    {
        int nr = 4;
        double[] r = new double[nr];
        int s = 123457;

        /* Initialize MersenneTwister with a seed */
        MersenneTwister mt1 = new MersenneTwister(s);
        MersenneTwister mt2 = (MersenneTwister) mt1.Clone();

        /* Save the state of MersenneTwister */
        Stream stm = new FileStream("mt", FileMode.Create);
        IFormatter fmt = new BinaryFormatter();
        fmt.Serialize(stm, mt1);
        stm.Flush();
        stm.Close();

        Imsl.Stat.Random rndm = new Imsl.Stat.Random(mt1);

        /* Get the next five random numbers */
        for (int k = 0; k < nr; k++)
        {
            r[k] = rndm.NextDouble();
        }
    }
}
```csharp
Console.WriteLine("First Stream Output");
Console.WriteLine(r[0] + " " + r[1] + " " + r[2] + " " + r[3]);

/* Check the cloned copy against the original */
for (int k=0; k < nr; k++)
{
    r[k] = rndm2.NextDouble();
}
Console.WriteLine("Clone Stream Output");
Console.WriteLine(r[0] + " " + r[1] + " " + r[2] + " " + r[3]);

/* Check the serialized copy against the original */
System.IO.Stream stm2 = new FileStream("mt", FileMode.Open);
IFormatter fmt2 = new BinaryFormatter();
mt2 = (MersenneTwister)fmt2.Deserialize(stm2);
stm2.Close();
Imsl.Stat.Random rndm3 = new Imsl.Stat.Random(mt2);
for (int k=0; k < nr; k++)
{
    r[k] = rndm3.NextDouble();
}
Console.WriteLine("Serialized Stream Output");
Console.WriteLine(r[0] + " " + r[1] + " " + r[2] + " " + r[3]);
}
}

Output

First Stream Output
0.434745062375441 0.352208853699267 0.0138511140830815 0.20914130914025

Clone Stream Output
0.434745062375441 0.352208853699267 0.0138511140830815 0.20914130914025

Serialized Stream Output
0.434745062375441 0.352208853699267 0.0138511140830815 0.20914130914025

MersenneTwister64 Class

A 64-bit Mersenne Twister generator.
MersenneTwister64 generates uniform pseudorandom 64-bit numbers with a period of $2^{19937} - 1$ and a 623-dimensional equidistribution property. See Matsumoto et al. 1998 for details.

---

Random Number Generation MersenneTwister64 • 1211
Since 64-bit numbers are generated, all of the bits of both `nextFloat` and `nextDouble` are pseudorandom.

The series of random numbers can be generated using a seed for initialization or by using an array of type `int`. One can also save the state of the generator at initialization to be re-used later.

This C# code was translated from the following C program.

A C-program for MT19937, with initialization improved 2002/1/26. Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using `init_genrand(seed)` or `init_by_array(init_key, key_length)`.

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Any feedback is very welcome.

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

email: m-mat@math.sci.hiroshima-u.ac.jp

**Constructors**

**MersenneTwister64**

```csharp
public MersenneTwister64(int seed)
```
Description
Initializes the 64-bit Mersenne Twister generator using a seed.

Parameter
  seed – An int which contains the seed that is used to initialize the 64-bit Mersenne Twister generator.

MersenneTwister64
public MersenneTwister64(ulong seed)

Description
Initializes the 64-bit Mersenne Twister generator using a seed.

Parameter
  seed – A ulong which represents the seed used to initialize the 64-bit Mersenne Twister generator.

MersenneTwister64
public MersenneTwister64(int[] key)

Description
Initializes the 64-bit Mersenne Twister generator with supplied array.

Parameter
  key – A int array used to initialize the 64-bit Mersenne Twister generator.

MersenneTwister64
public MersenneTwister64(ulong[] key)

Description
Initializes the 64-bit Mersenne Twister generator with supplied array.

Parameter
  key – A ulong array used to initialize the 64-bit Mersenne Twister generator.

Methods

Clone
Final public object Clone()

Description
Returns a clone of this object.

Returns
An Object which is a clone of this MersenneTwister64 object.

Next
virtual public int Next()
Description
Returns a nonnegative random number.

Returns
A 32-bit signed integer greater than or equal to zero.

**NextDouble**
virtual public double NextDouble()

Description
Returns a random number between 0.0 and 1.0.

Returns
A double greater than or equal to 0.0, and less than 1.0.

**NextFloat**
virtual public float NextFloat()

Description
Returns a random number between 0.0 and 1.0.

Returns
A float greater than or equal to 0.0, and less than 1.0.

**NextLong**
virtual public long NextLong()

Description
Generates the next pseudorandom, uniformly distributed long value from this random number generator’s sequence.

Returns
A long from this random number generator’s sequence.

**Example: Mersenne Twister Random Number Generation**
In this example, four simulation streams are generated. The first series is generated with the seed used for initialization. The second series is generated using an array for initialization. The third series is obtained by resetting the generator back to the state it had at the beginning of the second stream. Therefore, the second and third streams are identical. The fourth stream is obtained by resetting the generator back to its original, uninitialized state, and having it reinitialize using the seed. The first and fourth streams are therefore the same.

```csharp
using System;
using System.IO;
using System.Runtime.Serialization;
using Imsl.Stat;
```

1214 • MersenneTwister64 IMSL C# Numerical Library
public class MersenneTwister64Ex1
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    static void Main(string[] args)
    {
        int nr = 4;
        double[] r = new double[nr];
        int s = 123457;

        /* Initialize MersenneTwister64 with a seed */
        MersenneTwister64 mt1 = new MersenneTwister64(s);
        MersenneTwister64 mt2 = (MersenneTwister64)mt1.Clone();

        /* Save the state of MersenneTwister64 */
        Stream stm = new FileStream("mt", FileMode.Create);
        IFormatter fmt = new BinaryFormatter();
        fmt.Serialize(stm, mt1);
        stm.Flush();
        stm.Close();

        Imsl.Stat.Random rndm = new Imsl.Stat.Random(mt1);

        /* Get the next five random numbers */
        for (int k=0; k < nr; k++)
        {
            r[k] = rndm.NextDouble();
        }

        Console.WriteLine("First Stream Output");

        /* Check the cloned copy against the original */
        for (int k=0; k < nr; k++)
        {
            r[k] = rndm2.NextDouble();
        }

        Console.WriteLine("Clone Stream Output");

        /* Check the serialized copy against the original */
        System.IO.Stream stm2 = new FileStream("mt", FileMode.Open);
        IFormatter fmt2 = new BinaryFormatter();
        mt2 = (MersenneTwister64)fmt2.Deserialize(stm2);
        stm2.Close();

        Imsl.Stat.Random rndm3 = new Imsl.Stat.Random(mt2);
        for (int k=0; k < nr; k++)
        {
            r[k] = rndm3.NextDouble();
        }
    }
}
```csharp
Console.WriteLine("\nSerialized Stream Output");
}
}

Output

First Stream Output
0.579916541818503 0.940114746325065 0.710159376724905 0.163995293979278

Clone Stream Output
0.579916541818503 0.940114746325065 0.710159376724905 0.163995293979278

Serialized Stream Output
0.579916541818503 0.940114746325065 0.710159376724905 0.163995293979278

---

## FaureSequence Class

```csharp
public class Imsl.Stat.FaureSequence : Imsl.Stat.IRandomSequence

Generates the low-discrepancy Faure sequence.

Discrepancy measures the deviation from uniformity of a point set.

The discrepancy of the point set \(x_1, \ldots, x_n \in [0,1]^d, d \geq 1\), is

\[
D_n^{(d)} = \sup_E \left| \frac{A(E;n)}{n} - \lambda(E) \right|,
\]

where the supremum is over all subsets of \([0,1]^d\) of the form

\[E = [0,t_1) \times \cdots \times [0,t_d), \ 0 \leq t_j \leq 1, \ 1 \leq j \leq d,\]

\(\lambda\) is the Lebesgue measure, and \(A(E;n)\) is the number of the \(x_j\) contained in \(E\).

The sequence \(x_1, x_2, \ldots\) of points in \([0,1]^d\) is a low-discrepancy sequence if there exists a constant \(c(d)\), depending only on \(d\), such that

\[
D_n^{(d)} \leq c(d) \left(\frac{\log n}{n}\right)^d
\]

for all \(n > 1\).

Generalized Faure sequences can be defined for any prime base \(b \geq d\). The lowest bound for the discrepancy is obtained for the smallest prime \(b \geq d\), so the base defaults to the smallest prime greater than or equal to the dimension.

The generalized Faure sequence \(x_1, x_2, \ldots\) is computed as follows:
Write the positive integer $n$ in its $b$-ary expansion,

$$n = \sum_{i=0}^{\infty} a_i(n)b^i$$

where $a_i(n)$ are integers, $0 \leq a_i(n) < b$.

The $j$-th coordinate of $x_n$ is

$$x_n^{(j)} = \sum_{k=0}^{\infty} \sum_{d=0}^{\infty} c_{kd}^{(j)} d_d(n)b^{-k-1}, 1 \leq j \leq d$$

The generator matrix for the series, $c_{kd}^{(j)}$, is defined to be

$$c_{kd}^{(j)} = j^{d-k} c_{kd}$$

and $c_{kd}$ is an element of the Pascal matrix,

$$c_{kd} = \begin{cases} \frac{d!}{c!(d-c)!} & k \leq d \\ 0 & k > d \end{cases}$$

It is faster to compute a shuffled Faure sequence than to compute the Faure sequence itself. It can be shown that this shuffling preserves the low-discrepancy property.

The shuffling used is the $b$-ary Gray code. The function $G(n)$ maps the positive integer $n$ into the integer given by its $b$-ary expansion. The sequence computed by this function is $\vec{x}(G(n))$, where $\vec{x}$ is the generalized Faure sequence.

### Properties

<table>
<thead>
<tr>
<th>Base</th>
<th>public int Base {get; }</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The base.</td>
</tr>
<tr>
<td>Property Value</td>
<td>A int which specifies the base.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Final public int Dimension {get; }</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the dimension of the sequence.</td>
</tr>
</tbody>
</table>
Property Value
A \texttt{int} which specifies the dimension.

\textbf{NumberOfProcessors}
\begin{verbatim}
public int NumberOfProcessors {get; set; }
\end{verbatim}

Description
Perform the parallel calculations with the maximum possible number of processors set to 
\texttt{NumberOfProcessors}.

Property Value
An \texttt{int} indicating the maximum possible number of processors to use.

Remarks
By default, \texttt{NumberOfProcessors} = \texttt{Environment.ProcessorCount}. If \texttt{NumberOfProcessors} is
set to a number less than 1 or greater than \texttt{Environment.ProcessorCount},
\texttt{Environment.ProcessorCount} will be used internally. This property has no effect if the application is
used with a scalar version of the IMSL C# Numerical Library.

\textbf{Skip}
\begin{verbatim}
public int Skip {get; }
\end{verbatim}

Description
Returns the number of points skipped at the beginning of the sequence.

Property Value
A \texttt{int} which specifies the number of points skipped.

\section*{Constructors}

\textbf{FaureSequence}
\begin{verbatim}
public FaureSequence(int dimension)
\end{verbatim}

Description
Creates a Faure sequence with the default base.

Parameter
\begin{verbatim}
dimension – An int which specifies the dimension of the sequence.
\end{verbatim}

Remarks
The base defaults to the smallest prime equal to or greater than \texttt{dimension}.

\textbf{FaureSequence}
\begin{verbatim}
public FaureSequence(int dimension, int baseSequence, int nSkip)
\end{verbatim}
Description
Creates a Faure sequence.

Parameters
- dimension – An int which specifies the dimension of the sequence.
- baseSequence – A int which specifies the smallest prime number greater than or equal to dimension.
- nSkip – An int which specifies the number of initial points to skip.

Remarks
If nSkip is negative then \( base^{m/2−1} \), where \( m \) is the number of digits needed to represent the largest Int32 in the base, points are skipped.

Methods

ComputeParameters
void ComputeParameters()

Description
Compute needed parameters.

NextDouble
public double NextDouble()

Description
Returns the first value of the next point in the sequence.

Returns
A double array which specifies the next sequence value.

Remarks
This method is intended for use when dimension is 1.

NextPoint
Final public double[] NextPoint()

Description
Returns the next point in the sequence.

Returns
A double array which specifies the next point in the sequence.

NextPrime
static public int NextPrime(int n)
**Description**

Returns the smallest prime greater than or equal to n.

**Parameter**

n – An int which specifies the first number to try as a prime.

**Returns**

An int which specifies a prime greater than or equal to n.

**Remarks**

If n is less than or equal to 2 then 2 is returned.

**Example: FaureSequence**

In this example, ten points of the Faure sequence are computed. The points are in a four-dimensional cube.

```csharp
using System;
using FaureSequence = Imsl.Stat.FaureSequence;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class FaureSequenceEx1
{
    public static void Main(String[] args)
    {
        FaureSequence seq = new FaureSequence(4);
        double[,] x = new double[10, ];
        for (int k = 0; k < 10; k++)
        {
            x[k] = seq.NextPoint();
        }
        new PrintMatrix("Faure Sequence").Print(x);
    }
}
```

**Output**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.201344</td>
<td>0.274944</td>
<td>0.532544</td>
</tr>
<tr>
<td>1</td>
<td>0.401344</td>
<td>0.474944</td>
<td>0.732544</td>
</tr>
<tr>
<td>2</td>
<td>0.601344</td>
<td>0.674944</td>
<td>0.932544</td>
</tr>
<tr>
<td>3</td>
<td>0.801344</td>
<td>0.874944</td>
<td>0.132544</td>
</tr>
<tr>
<td>4</td>
<td>0.841344</td>
<td>0.114944</td>
<td>0.572544</td>
</tr>
<tr>
<td>5</td>
<td>0.041344</td>
<td>0.314944</td>
<td>0.772544</td>
</tr>
<tr>
<td>6</td>
<td>0.241344</td>
<td>0.514944</td>
<td>0.972544</td>
</tr>
<tr>
<td>7</td>
<td>0.441344</td>
<td>0.714944</td>
<td>0.172544</td>
</tr>
<tr>
<td>8</td>
<td>0.641344</td>
<td>0.914944</td>
<td>0.372544</td>
</tr>
<tr>
<td>9</td>
<td>0.681344</td>
<td>0.154944</td>
<td>0.612544</td>
</tr>
</tbody>
</table>

1220 • FaureSequence

IMSL C# Numerical Library
IRandomSequence Interface

public interface Imsl.Stat.IRandomSequence
Interface implemented by generators of random or quasi-random multidimension sequences.

Property

**Dimension**
abstract public int Dimension {get; }

**Description**
Returns the dimension of the sequence.

**Property Value**
A int which specifies the dimension.

Method

**NextPoint**
abstract public double[] NextPoint()

**Description**
Returns the next multidimensional point in the sequence.

**Returns**
A double array of length *dimension*. 
Chapter 23: Finance

Types

\textit{class} Finance .......................................................... 1224  
\textit{enumeration} Finance.Period ........................................ 1254  
\textit{class} Bond .............................................................. 1254  
\textit{enumeration} Bond.Frequency ....................................... 1288  
\textit{class} DayCountBasis .................................................. 1289  
\textit{interface} IBasisPart ............................................... 1291

Usage Notes

Users can perform financial computations by using pre-defined data types. Most of the financial functions require one or more of the following:

- Date
- Number of payments per year
- A variable to indicate when payments are due
- Day count basis

The Bond.Frequency field indicates the number of payments for each year.

<table>
<thead>
<tr>
<th>Bond.Frequency</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bond.Annual</td>
<td>One payment per year (Annual payment)</td>
</tr>
<tr>
<td>Bond.SemiAnnual</td>
<td>Two payments per year (Semi-annual payment)</td>
</tr>
<tr>
<td>Bond.Quarterly</td>
<td>Four payments per year (Quarterly payment)</td>
</tr>
</tbody>
</table>

The Finance.Period field indicates when payments are due.

<table>
<thead>
<tr>
<th>Finance.Period</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finance.At_End_of_Period</td>
<td>Payments are due at the end of the period</td>
</tr>
<tr>
<td>Finance.AT_Beginning_of_Period</td>
<td>Payments are due at the beginning of the period</td>
</tr>
</tbody>
</table>
The `DayCountBasis` class provides fields to indicate the type of day count basis. Day count basis is the method for computing the number of days between two dates.

<table>
<thead>
<tr>
<th>Class Field</th>
<th>Day count basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>DayCountBasis.BasisNASD</td>
<td>US (NASD) 30/360</td>
</tr>
<tr>
<td>DayCountBasis.BasisActualActual</td>
<td>Actual/Actual</td>
</tr>
<tr>
<td>DayCountBasis.BasisActual360</td>
<td>Actual/360</td>
</tr>
<tr>
<td>DayCountBasis.BasisActual365</td>
<td>Actual/365</td>
</tr>
<tr>
<td>DayCountBasis.Basis30e360</td>
<td>European 30/360</td>
</tr>
</tbody>
</table>

**Additional Information**

In preparing the finance and bond functions we incorporated standards used by *SIA Standard Securities Calculation Methods*.

More detailed information on finance and bond functionality can be found in the following manuals:

- *Microsoft Excel 5, Worksheet Function Reference*.

---

**Finance Class**

```csharp
public class Imsl.Finance.Finance

Collection of finance functions.
```

**Constructor**

```csharp
public Finance()

Description
Initializes a new instance of the Imsl.Finance.Finance (p. 1224) class.
```

**Methods**

```csharp
static public double Cumipmt(double rate, int nper, double pv, int firstPeriod, int lastPeriod, Imsl.Finance.Finance.Period period)
```

**Finance IMSL C# Numerical Library**
Description
Returns the cumulative interest paid between two periods.

Parameters
rate – A double which specifies the interest rate.
nper – A int which specifies the total number of payment periods.
pv – A double which specifies the present value.
firstPeriod – A int containing the first period in the calculation. Periods are numbered starting with one.
lastPeriod – A int which specifies the last period in the calculation.
period – A int which specifies the time in each period when the payment is made, either Imsl.Finance.Finance.Period.AtEnd (p. 1254) or Imsl.Finance.Finance.Period.AtBeginning (p. 1254)

Returns
A double which specifies the cumulative interest paid between the first period and the last period.

Remarks
It is computed using the following:

\[ \sum_{i=\text{firstPeriod}}^{\text{lastPeriod}} \text{interest}_i \]

where \( \text{interest}_i \) is computed from Ipmt for the \( i \)-th period.

Cumprinc
static public double Cumprinc(double rate, int nper, double pv, int firstPeriod, int lastPeriod, Imsl.Finance.Finance.Period time)

Description
Returns the cumulative principal paid between two periods.

Parameters
rate – A double which specifies the interest rate.
nper – A int which specifies the total number of payment periods.
pv – A double which specifies the present value.
firstPeriod – A int which specifies the first period in the calculation. Periods are numbered starting with one.
lastPeriod – A int which specifies the last period in the calculation.
time – The time of a Period when the payment is made (either Imsl.Finance.Finance.Period.AtEnd (p. 1254) or Imsl.Finance.Finance.Period.AtBeginning (p. 1254)).

Returns
A double which specifies the cumulative principal paid between the first period and the last period.
Remarks
It is computed using the following:

\[ \sum_{i={\text{firstPeriod}}}^{\text{lastPeriod}} \text{principal}_i \]

where \( \text{principal}_i \) is computed from \( \text{Pmt} \) for the \( i \)-th period.

**Db**

static public double Db(double cost, double salvage, int life, int period, int month)

**Description**
Returns the depreciation of an asset using the fixed-declining balance method.

**Parameters**
- `cost` - A double which specifies the initial cost of the asset.
- `salvage` - A double which specifies the salvage value of the asset.
- `life` - A int which specifies the number of periods over which the asset is being depreciated.
- `period` - A int which specifies the period for which the depreciation is to be computed.
- `month` - A int which specifies the number of months in the first year.

**Returns**
A double which specifies the depreciation of an asset for a specified period using the fixed-declining balance method.

**Remarks**
Method Db varies depending on the specified value for the argument period, see table below.

If period = 1,

\[ \text{cost} \times \text{rate} \times \frac{\text{month}}{12} \]

If period = life,

\[ (\text{cost} - \text{total depreciation from periods}) \times \text{rate} \times \frac{12 - \text{month}}{12} \]

If period other than 1 or life,

\[ (\text{cost} - \text{total depreciation from prior periods}) \times \text{rate} \]

where

\[ \text{rate} = 1 - \left( \frac{\text{salvage}}{\text{cost}} \right)^{\left( \frac{\text{life}}{12} \right)} \]

NOTE: rate is rounded to three decimal places.
**Ddb**

```java
static public double Ddb(double cost, double salvage, int life, int period, double factor)
```

**Description**

Returns the depreciation of an asset using the double-declining balance method.

**Parameters**

- `cost` – A `double` which specifies the initial cost of the asset.
- `salvage` – A `double` which specifies the salvage value of the asset.
- `life` – A `int` which specifies the number of periods over which the asset is being depreciated.
- `period` – A `int` which specifies the period.
- `factor` – A `double` which specifies the rate at which the balance declines.

**Returns**

A `double` which specifies the depreciation of an asset for a specified period.

**Remarks**

It is computed using the following:

\[
\frac{\text{cost} - \text{salvage} \times \left(\text{total depreciation from prior periods}\right)}{\text{life}} \times \frac{\text{factor}}{\text{life}}
\]

---

**Dollarde**

```java
static public double Dollarde(double fractionalDollar, int fraction)
```

**Description**

Converts a fractional price to a decimal price.

**Parameters**

- `fractionalDollar` – A `double` which specifies a fractional number.
- `fraction` – A `int` which specifies the denominator.

**Returns**

A `double` which specifies the dollar price expressed as a decimal number.

**Remarks**

It is computed using the following:

\[
idollar + \left(\text{fractionalDollar} - idollar\right) \times \frac{10^{(ifrac+1)}}{fraction}
\]

where `idollar` is the integer part of `fractionalDollar`, and `ifrac` is the integer part of \(\log(fraction)\).
Description
Converts a decimal price to a fractional price.

Parameters
- `decimalDollar` – A `double` which specifies a decimal number.
- `fraction` – A `int` which specifies the denominator.

Returns
A `double` which specifies a dollar price expressed as a fraction.

Remarks
It is computed using the following:

\[ idollar + \frac{decimalDollar - idollar}{10^{ifrac+1}/fraction} \]

where `idollar` is the integer part of the `decimalDollar`, and `ifrac` is the integer part of \( \log(fraction) \).

Effect
static public double Effect(double nominalRate, int nper)

Description
Returns the effective annual interest rate.

Parameters
- `nominalRate` – A `double` which specifies the nominal interest rate.
- `nper` – A `int` which specifies the number of compounding periods per year.

Returns
A `double` which specifies the effective annual interest rate.

Remarks
The nominal interest rate is the periodically-compounded interest rate as stated on the face of a security. The effective annual interest rate is computed using the following:

\[ \left( 1 + \frac{nominalRate}{nper} \right)^{nper} - 1 \]

Fv
static public double Fv(double rate, int nper, double pmt, double pv, Imsl.Finance.Finance.Period period)

Description
Returns the future value of an investment.
Parameters

rate – A double which specifies the interest rate.
nper – A int which specifies the total number of payment periods.
pmt – A double which specifies the payment made in each period.
pv – A double which specifies the present value.
period – A int which specifies the time in each period when the payment is made (either Imsl.Finance.Finance.Period.AtEnd (p. 1254) or Imsl.Finance.Finance.Period.AtBeginning (p. 1254)).

Returns

A double which specifies the future value of an investment.

Remarks

The future value is the value, at some time in the future, of a current amount and a stream of payments. It can be found by solving the following:

If $rate = 0$,

$$pv + pmt \times nper + fv = 0$$

If $rate \neq 0$,

$$pv(1 + rate)^{nper} + pmt(1 + rate\times\text{period})\left(\frac{(1 + rate)^{nper} - 1}{rate}\right) + fv = 0$$

Fvschedule

static public double Fvschedule(double principal, double[] schedule)

Description

Returns the future value of an initial principal taking into consideration a schedule of compound interest rates.

Parameters

principal – A double which specifies the present value.
schedule – A double array of interest rates to apply.

Returns

A double which specifies the future value of an initial principal

Remarks

It is computed using the following:

$$\sum_{i=1}^{\text{count}} (\text{principal} \times \text{schedule}_i)$$

where $\text{schedule}_i = \text{interest rate at the } i\text{-th period.}$
Ipmt
static public double Ipmt(double rate, int period, int nper, double pv, double fv, Imsl.Finance.Finance.Period time)

Description
Returns the interest payment for an investment for a given period.

Parameters
- rate – A double which specifies the interest rate.
- period – A int which specifies the payment period.
- nper – A int which specifies the total number of periods.
- pv – A double which specifies the present value.
- fv – A double which specifies the future value.
- time – The time of a Period when the payment is made (either Imsl.Finance.Finance.Period.AtEnd (p. 1254) or Imsl.Finance.Finance.Period.AtBeginning (p. 1254)).

Returns
A double which specifies the interest payment for a given period for an investment.

Remarks
It is computed using the following:
\[
\left\{ pv (1 + rate)^{nper - 1} + pmt (1 + rate \times period) \frac{(1 + rate)^{nper - 1}}{rate} \right\} rate
\]

Irr
static public double Irr(double[] pmt)

Description
Returns the internal rate of return for a schedule of cash flows.

Parameter
- pmt – A double array which contains cash flow values which occur at regular intervals.

Returns
A double which specifies the internal rate of return.

Remarks
It is found by solving the following:
\[
0 = \sum_{i=1}^{\text{count}} \frac{value_i}{(1 + rate)^i}
\]

where \(value_i\) is the \(i\)th cash flow, \(rate\) is the internal rate of return.
**Irr**

static public double Irr(double[] pmt, double guess)

**Description**

Returns the internal rate of return for a schedule of cash flows.

**Parameters**

- `pmt` – A double array which contains cash flow values which occur at regular intervals.
- `guess` – A double value which represents an initial guess at the return value from this function.

**Returns**

A double which specifies the internal rate of return.

**Remarks**

It is found by solving the following:

\[ 0 = \sum_{i=1}^{\text{count}} \frac{\text{value}_i}{(1 + \text{rate})^i} \]

where \( \text{value}_i \) = the \( i \)th cash flow, \( \text{rate} \) is the internal rate of return.

**Mirr**

static public double Mirr(double[] cashFlow, double financeRate, double reinvestRate)

**Description**

Returns the modified internal rate of return for a schedule of periodic cash flows.

**Parameters**

- `cashFlow` – A double array of cash flows.
- `financeRate` – A double which specifies the interest you pay on the money you borrow.
- `reinvestRate` – A double which specifies the interest rate you receive on the cash flows.

**Returns**

A double which specifies the modified internal rate of return.

**Remarks**

The modified internal rate of return differs from the ordinary internal rate of return in assuming that the cash flows are reinvested at the cost of capital, not at the internal rate of return. It also eliminates the multiple rates of return problem. It is computed using the following:

\[
\left\{ \frac{- (\text{npv}) (1 + \text{reinvestRate})^{n\text{per}}}{(\text{npv}) (1 + \text{financeRate})} \right\}^{\frac{1}{n\text{per}}} - 1
\]

where \( \text{npv} \) is calculated from \( \text{Npv} \) for positive values in \( \text{values} \) using \( \text{reinvestRate} \), and where \( \text{nnpv} \) is calculated from \( \text{Npv} \) for negative values in \( \text{values} \) using \( \text{financeRate} \).
**Nominal**

```csharp
static public double Nominal(double effectiveRate, int nper)
```

**Description**

Returns the nominal annual interest rate.

**Parameters**

- `effectiveRate` – A `double` which specifies the effective interest rate.
- `nper` – A `int` which specifies the number of compounding periods per year.

**Returns**

A `double` which specifies the nominal annual interest rate.

**Remarks**

The nominal interest rate is the interest rate as stated on the face of a security. It is computed using the following:

\[
\left[(1 + \text{effectiveRate})^{\frac{1}{nper}} - 1\right] \times nper
\]

---

**Nper**

```csharp
static public double Nper(double rate, double pmt, double pv, double fv, Imsl.Finance.Finance.Period period)
```

**Description**

Returns the number of periods for an investment for which periodic, and constant payments are made and the interest rate is constant.

**Parameters**

- `rate` – A `double` which specifies the interest rate.
- `pmt` – A `double` which specifies the payment.
- `pv` – A `double` which specifies the present value.
- `fv` – A `double` which specifies the future value.
- `period` – A `int` which specifies the time in each period when the payment is made (either `Imsl.Finance.Finance.Period.AtEnd` (p. 1254) or `Imsl.Finance.Finance.Period.AtBeginning` (p. 1254)).

**Returns**

A `int` which specifies the number of periods for an investment.
### Remarks
It can be found by solving the following:

If \( \text{rate} = 0 \),
\[
    pv + pmt \times nper + fv = 0
\]

If \( \text{rate} \neq 0 \),
\[
    pv(1 + rate)^{nper} + pmt[1 + rate \times \text{period}] \times \frac{(1 + rate)^{nper} - 1}{rate} + fv = 0
\]

---

### Npv

static public double Npv(double rate, double[] eqCashFlow)

**Description**

Returns the net present value of a stream of equal periodic cash flows, which are subject to a given discount rate.

**Parameters**

- rate – A double which specifies the interest rate per period.
- eqCashFlow – A double array of equally-spaced cash flows.

**Returns**

A double which specifies the net present value of the investment.

**Remarks**

It is found by solving the following:

\[
    \sum_{i=1}^{\text{count}} \frac{value_i}{(1 + rate)^i}
\]

where \( value_i \) = the \( i \)th cash flow.

---

### PeriodicPayment

static public double PeriodicPayment(double rate, int nper, double pv, double fv, Imsl.Finance.Finance.Period period)

**Description**

Returns the periodic payment for an investment.

**Parameters**

- rate – A double which specifies the interest rate.
- nper – A int which specifies the total number of periods.
- pv – A double which specifies the present value.
- fv – A double which specifies the future value.
- period – A int which specifies the time in each period when the payment is made (either Imsl.Finance.Finance.Period.AtEnd (p. 1254) or Imsl.Finance.Finance.Period.AtBeginning (p. 1254)).
Returns
A double which specifies the interest payment for a given period for an investment.

Remarks
It can be found by solving the following:
If \( \text{rate} = 0 \),
\[
\text{pv} + \text{pmt} \times \text{nper} + \text{fv} = 0
\]
If \( \text{rate} \neq 0 \),
\[
\text{pv}(1 + \text{rate})^\text{nper} + \text{pmt}[1 + \text{rate} \times \text{period}]\left(\frac{(1 + \text{rate})^\text{nper} - 1}{\text{rate}}\right) + \text{fv} = 0
\]

Ppmt
static public double Ppmt(double rate, int period, int nper, double pv, double fv, Imsl.Finance.Finance.Period time)

Description
Returns the payment on the principal for a specified period.

Parameters
- rate – A double which specifies the interest rate.
- period – A int which specifies the payment period.
- nper – A int which specifies the total number of periods.
- pv – A double which specifies the present value.
- fv – A double which specifies the future value.
- time – The time of a Period when the payment is made (either Imsl.Finance.Finance.Period.AtEnd (p. 1254) or Imsl.Finance.Finance.Period.AtBeginning (p. 1254)).

Returns
A double which specifies the payment on the principal for a given period.

Remarks
It is computed using the following:
\[
\text{payment}_i - \text{interest}_i
\]
where \( \text{payment}_i \) is computed from \( \text{pmt} \) for the \( i \)-th period, \( \text{interest}_i \) is calculated from \( \text{Ipmt} \) for the \( i \)-th period.

Pv
static public double Pv(double rate, int nper, double pmt, double fv, Imsl.Finance.Finance.Period time)
Description
Returns the net present value of a stream of equal periodic cash flows, which are subject to a given discount rate.

Parameters
- `rate` - A `double` which specifies the interest rate per period.
- `nper` - A `int` which specifies the number of periods.
- `pmt` - A `double` which specifies the payment made each period.
- `fv` - A `double` which specifies the annuity’s value after the last payment.
- `time` - The time in a `Period` when the payment is made (either `Imsl.Finance.Finance.Period.AtEnd` (p. 1254) or `Imsl.Finance.Finance.Period.AtBeginning` (p. 1254)).

Returns
A `double` which specifies the present value of the investment.

Remarks
It can be found by solving the following:
If `rate = 0`,

\[ pv + pmt \times nper + fv = 0 \]

If `rate ≠ 0`,

\[ pv(1 + rate)^{nper} + pmt[1 + rate (period)] \frac{(1 + rate)^{nper} - 1}{rate} + fv = 0 \]

Rate
static public double Rate(int nper, double pmt, double pv, double fv, Imsl.Finance.Finance.Period time)

Description
Returns the interest rate per period of an annuity.

Parameters
- `nper` - A `int` which specifies the number of periods.
- `pmt` - A `double` which specifies the payment made each period.
- `pv` - A `double` which specifies the present value.
- `fv` - A `double` which specifies the annuity’s value after the last payment.
- `time` - The time in a `Period` when the payment is made (either `Imsl.Finance.Finance.Period.AtEnd` (p. 1254) or `Imsl.Finance.Finance.Period.AtBeginning` (p. 1254)).

Returns
A `double` which specifies the interest rate per period of an annuity.
Remarks
Rate is calculated by iteration and can have zero or more solutions. It can be found by solving the following:
If \( rate = 0 \),
\[
pv + pmt \times nper + fv = 0
\]
If \( rate \neq 0 \),
\[
pv(1 + rate)^{nper} + pmt[1 + rate (period)]\left(\frac{(1 + rate)^{nper} - 1}{rate}\right) + fv = 0
\]

Rate
static public double Rate(int nper, double pmt, double pv, double fv, Imsl.Finance.Finance.Period time, double guess)

Description
Returns the interest rate per period of an annuity with an initial guess.

Parameters
- nper – A int which specifies the number of periods.
- pmt – A double which specifies the payment made each period.
- pv – A double which specifies the present value.
- fv – A double which specifies the annuity’s value after the last payment.
- time – The time in a Period when the payment is made (either Imsl.Finance.Finance.Period.AtEnd (p. 1254) or Imsl.Finance.Finance.Period.AtBeginning (p. 1254)).
- guess – A double value which represents an initial guess at the interest rate per period of an annuity.

Returns
A double which specifies the interest rate per period of an annuity.

Remarks
Rate is calculated by iteration and can have zero or more solutions. It can be found by solving the following:
If \( rate = 0 \),
\[
pv + pmt \times nper + fv = 0
\]
If \( rate \neq 0 \),
\[
pv(1 + rate)^{nper} + pmt[1 + rate (period)]\left(\frac{(1 + rate)^{nper} - 1}{rate}\right) + fv = 0
\]

Sln
static public double Sln(double cost, double salvage, int life)
**Description**
Returns the depreciation of an asset using the straight line method.

**Parameters**
- `cost` – A `double` which specifies the initial cost of the asset.
- `salvage` – A `double` which specifies the salvage value of the asset.
- `life` – A `int` which specifies the number of periods over which the asset is being depreciated.

**Returns**
A `double` which specifies the straight line depreciation of an asset for one period.

**Remarks**
It is computed using the following:

\[ \frac{\text{cost} - \text{salvage}}{\text{life}} \]

---

**Syd**
static public double Syd(double cost, double salvage, int life, int per)

**Description**
Returns the depreciation of an asset using the sum-of-years digits method.

**Parameters**
- `cost` – A `double` which specifies the initial cost of the asset.
- `salvage` – A `double` which specifies the salvage value of the asset.
- `life` – A `int` which specifies the number of periods over which the asset is being depreciated.
- `per` – A `int` which specifies the period.

**Returns**
A `double` which specifies the sum-of-years digits depreciation of an asset.

**Remarks**
It is computed using the following:

\[ \left( \frac{\text{cost} - \text{salvage}}{\text{life}} \right) \frac{(\text{life} + 1)(\text{life})}{2} \]

---

**Vdb**
static public double Vdb(double cost, double salvage, int life, int firstPeriod, int lastPeriod, double factor, bool noSL)

**Description**
Returns the depreciation of an asset for any given period using the variable-declining balance method.
Parameters

cost – A double which specifies the initial cost of the asset.
salvage – A double which specifies the salvage value of the asset.
life – A int which specifies the number of periods over which the asset is being depreciated.
firstPeriod – A int which specifies the first period for the calculation.
lastPeriod – A int which specifies the last period for the calculation.
factor – A double which specifies the rate at which the balance declines.
noSL – A bool flag. If true, do not switch to straight-line depreciation even when the depreciation is greater than the declining balance calculation.

Returns
A double which specifies the depreciation of the asset.

Remarks
It is computed using the following:
If no_SL = 0,
\[
\sum_{i=\text{firstPeriod}+1}^{\text{lastPeriod}} ddb_i
\]
If no_SL ≠ 0,
\[
A + \sum_{i=k}^{\text{lastPeriod}} \frac{\text{cost} - A - \text{salvage}}{\text{lastPeriod} - k + 1}
\]
where \( ddb_i \) is computed from \( Ddb \) for the \( i \)-th period. \( k = \) the first period where straight line depreciation is greater than the depreciation using the double-declining balance method.

\[
A = \sum_{i=\text{firstPeriod}+1}^{k-1} ddb_i
\]

Xirr

\[
\text{Xirr(double[]} \ pmt, \ System.DateTimem[] \ dates)\]

Description
Returns the internal rate of return for a schedule of cash flows.

Parameters
pmt – A double array which contains cash flow values which correspond to a schedule of payments in dates.
dates – A DateTime array which contains a schedule of payment dates.

Returns
A double which specifies the internal rate of return.
Remarks
It is not necessary that the cash flows be periodic. It can be found by solving the following:

\[ 0 = \sum_{i=1}^{\text{count}} \frac{\text{value}_i}{(1 + \text{rate})^{\frac{d_i - d_1}{365}}} \]

In the equation above, \( d_i \) represents the \( i \)th payment date. \( d_1 \) represents the 1st payment date. \( \text{value}_i \) represents the \( i \)th cash flow. \( \text{rate} \) is the internal rate of return.

Xirr
static public double Xirr(double[] pmt, System.DateTime[] dates, double guess)

Description
Returns the internal rate of return for a schedule of cash flows with a user supplied initial guess.

Parameters
- \( \text{pmt} \) – A double array which contains cash flow values which correspond to a schedule of payments in dates.
- \( \text{dates} \) – A DateTime array which contains a schedule of payment dates.
- \( \text{guess} \) – A double value which represents an initial guess at the return value from this function.

Returns
A double which specifies the internal rate of return.

Remarks
It is not necessary that the cash flows be periodic. It can be found by solving the following:

\[ 0 = \sum_{i=1}^{\text{count}} \frac{\text{value}_i}{(1 + \text{rate})^{\frac{d_i - d_1}{365}}} \]

In the equation above, \( d_i \) represents the \( i \)th payment date. \( d_1 \) represents the 1st payment date. \( \text{value}_i \) represents the \( i \)th cash flow. \( \text{rate} \) is the internal rate of return.

Xnpv
static public double Xnpv(double rate, double[] cashFlow, System.DateTime[] dates)

Description
Returns the present value for a schedule of cash flows.

Parameters
- \( \text{rate} \) – A double which specifies the interest rate.
- \( \text{cashFlow} \) – A double array containing the cash flows.
- \( \text{dates} \) – A DateTime array which contains a schedule of payment dates.
**Returns**
A double which specifies the present value.

**Remarks**
It is not necessary that the cash flows be periodic. It is computed using the following:

\[
\sum_{i=1}^{\text{count}} \frac{\text{value}_i}{(1 + \text{rate})^{(d_i - d_1)/365}}
\]

In the equation above, \(d_i\) represents the \(i\)th payment date, \(d_1\) represents the first payment date, and \(\text{value}_i\) represents the \(i\)th cash flow.

**Example: Cumulative Interest Example**
The amount of interest paid in the first year of a 30 year fixed rate mortgage is computed. The amount financed is $200,000 at an interest rate of 7.25% for 30 years.

```csharp
using System;
using Imsl.Finance;

public class cumipmtEx1
{
    public static void Main(String[] args)
    {
        double rate = 0.0725 / 12;
        int periods = 12 * 30;
        double pv = 200000;
        int start = 1;
        int end = 12;
        double total = Finance.Cumipmt(rate, periods, pv, start, end,
                                        Finance.Period.AtEnd);
        Console.Out.WriteLine("First year interest = " +
                             total.ToString("C");
    }
}
```

**Output**
First year interest = ($14,436.52)

**Example: Cumulative Principal Example**
The amount of principal paid in the first year of a 30 year fixed rate mortgage is computed. The amount financed is $200,000 at an interest rate of 7.25% for 30 years.

```csharp
using System;
using Imsl.Finance;

public class cumprincEx1
```
public static void Main(String[] args)
{
    double rate = 0.0725 / 12;
    int periods = 12 * 30;
    double pv = 200000;
    int start = 1;
    int end = 12;
    double total = Finance.Cumprinc(rate, periods, pv, start, end,
    Finance.Period.AtEnd);
    Console.Out.WriteLine("First year principal = "+
    total.ToString("C"));
}

Output
First year principal = ($1,935.71)

Example: Depreciation - Fixed Declining Balance Method

The depreciation of an asset with an initial cost of $2500 and a salvage value of $500 over a period of 3 years is calculated. Here month is 6 since the life of the asset did not begin until the seventh month of the first year.

using System;
using Imsl.Finance;

public class dbEx1
{
    public static void Main(String[] args)
    {
        double cost = 2500;
        double salvage = 500;
        int life = 3;
        int month = 6;

        for (int period = 1; period <= life + 1; period++)
        {
            double db = Finance.Db(cost, salvage, life, period, month);
            Console.Out.WriteLine("For period " + period + " "+
            db.ToString("C"));
        }
    }
}

Output
For period 1 $518.75
For period 2 $822.22
For period 3 $481.00
For period 4 $140.69
Example: Depreciation - Double-Declining Balance Method

The depreciation of an asset with an initial cost of $2500 and a salvage value of $500 over a period of 2 years is calculated. A factor of 2 is used (the double-declining balance method).

```csharp
using System;
using Imsl.Finance;

public class ddbEx1
{
    public static void Main(String[] args)
    {
        double cost = 2500;
        double salvage = 500;
        double factor = 2;
        int life = 24;

        for (int period = 1; period <= life; period++)
        {
            double ddb = Finance.Ddb(cost, salvage, life, period, factor);
            Console.Out.WriteLine("For period "+ period + " ddb = " + ddb.ToString("C"));
        }
    }
}
```

Output

For period 1  ddb = $208.33
For period 2  ddb = $190.97
For period 3  ddb = $175.06
For period 4  ddb = $160.47
For period 5  ddb = $147.10
For period 6  ddb = $134.84
For period 7  ddb = $123.60
For period 8  ddb = $113.30
For period 9  ddb = $103.86
For period 10  ddb = $95.21
For period 11  ddb = $87.27
For period 12  ddb = $80.00
For period 13  ddb = $73.33
For period 14  ddb = $67.22
For period 15  ddb = $61.62
For period 16  ddb = $56.48
For period 17  ddb = $51.78
For period 18  ddb = $47.46
For period 19  ddb = $22.09
For period 20  ddb = $0.00
For period 21  ddb = $0.00
For period 22  ddb = $0.00
For period 23  ddb = $0.00
For period 24  ddb = $0.00
Example: Price Conversion - Fractional Dollars

A fractional dollar price, in this case 1 3/8, is converted to a decimal price.

```csharp
using System;
using Imsl.Finance;

public class dollardeEx1
{
    public static void Main(String[] args)
    {
        double fractionalDollar = 1.3;
        int fraction = 8;
        double dollardec = Finance.Dollarde(fractionalDollar, fraction);
        Console.Out.WriteLine("The fractional dollar 1.3 = "+
            dollardec.ToString("C");
    }
}
```

Output
The fractional dollar 1.3 = $1.38

Example: Price Conversion - Decimal Dollars

A decimal dollar price, in this case $1.38, is converted to a fractional price.

```csharp
using System;
using Imsl.Finance;

public class dollarfrEx1
{
    public static void Main(String[] args)
    {
        double decimalDollar = 1.38;
        int fraction = 8;
        double dollarfrc = Finance.Dollarfr(decimalDollar, fraction);
        Console.Out.WriteLine("The decimal dollar $1.38 as a fractional"+
            " dollar = "+
            dollarfrc.ToString("0.00");
    }
}
```

Output
The decimal dollar $1.38 as a fractional dollar = 1.30

Example: Effective Rate

In this example the effective interest rate is computed given that the nominal rate is 6.0% and that the interest will be compounded quarterly.
```csharp
using System;
using Imsl.Finance;

class effectEx1
{
    public static void Main(String[] args)
    {
        double nominalRate = .06;
        int nper = 4;
        double effectiveRate;
        effectiveRate = Finance.Effect(nominalRate, nper);
        Console.Out.WriteLine("The effective rate of the nominal rate, 6.0%, compounded quarterly is "+ effectiveRate.ToString("P");
    }
}

Output
The effective rate of the nominal rate, 6.0%, compounded quarterly is 6.14 %

Example: Future Value of an Investment

A couple starts setting aside $30,000 a year when they are 45 years old. They expect to earn 5% interest on the money compounded yearly. The future value of the investment is computed for a 20 year period.

class fvEx1
{
    public static void Main(String[] args)
    {
        double rate = .05;
        int nper = 20;
        double payment = -30000.00;
        double pv = -30000.00;
        double fv = Finance.Fv(rate, nper, payment, pv, Finance.Period.AtBeginning);
        Console.Out.WriteLine("After 20 years, the value of the investments will be "+ fv.ToString("C");
    }
}

Output
After 20 years, the value of the investments will be $1,121,176.49
```
Example: Future Value - Adjustable Rates

An investment of $10,000 is made. The investment will grow at the rate of 5.1% the first year, with the rate increasing by .1% each year thereafter for a total of 5 years. The future value of the investment is computed.

```csharp
using System;
using Imsl.Finance;

public class fvscheduleEx1
{
    public static void Main(String[] args)
    {
        double principal = 10000.0;
        double[] schedule = new double[]{.050, .051, .052, .053, .054};
        double fvschedule;
        fvschedule = Finance.Fvschedule(principal, schedule);
        Console.Out.WriteLine("After 5 years the $10,000 investment " +
            "will have " + "grown to " +
            fvschedule.ToString("C");
    }
}
```

Output

After 5 years the $10,000 investment will have grown to $12,884.77

Example: Interest Payments

The interest due the second year on a $100,000 25 year loan is calculated. The loan is at 8%.

```csharp
using System;
using Imsl.Finance;

public class ipmtEx1
{
    public static void Main(String[] args)
    {
        double rate = .08;
        int per = 2;
        int nper = 25;
        double pv = 100000.00;
        double fv = 0.0;

        double ipmt = Finance.Ipmt(rate, per, nper, pv, fv,
                                    Finance.Period.AtEnd);
        Console.Out.WriteLine("The interest due the second year on the" +
            "$100,000 loan is " +
            ipmt.ToString("C");
    }
}
```
Output

The interest due the second year on the $100,000 loan is ($7,890.57)

Example: Internal Rate of Return

A farmer buys 10 young cows and a bull for $4500. The first year he does not expect to sell any calves, he just expects to feed them. Thereafter, he expects to be able to sell calves to offset the cost of feed. He expects them to be productive for 9 years, after which time he will liquidate the herd. The internal rate of return is computed after 9 years.

```csharp
using System;
using Imsl.Finance;

public class irrEx1
{
    public static void Main(String[] args)
    {
        double[] pmt = new double[]
        {- 4500.0, - 800.0,
         800.0, 800.0,
         600.0, 600.0,
         800.0, 800.0,
         700.0, 3000.0};

        double irr = Finance.Irr(pmt);
        Console.Out.WriteLine("After 9 years, the internal rate of return on the cows is "+
            irr.ToString("P");
    }
}
```

Output

After 9 years, the internal rate of return on the cows is 7.21 %

Example: Modified Internal Rate of Return

A farmer uses a $4500 loan to buy 10 young cows and a bull. The interest rate on the loan is 8%. He expects to reinvest the profits received in any one year in the money market and receive 5.5%. The first year he does not expect to sell any calves, he just expects to feed them. Thereafter, he expects to be able to sell calves to offset the cost of feed. He expects them to be productive for 9 years, after which time he will liquidate the herd. The modified internal rate of return is computed after 9 years.

```csharp
using System;
using Imsl.Finance;

public class mirrEx1
{
    public static void Main(String[] args)
    {
        double[] x = new double[]{- 4500.0, - 800.0,
```

1246 • Finance

IMSL C# Numerical Library
double financeRate = .08;
double reinvestRate = .055;
double mirr = Finance.Mirr(x, financeRate, reinvestRate);

Console.Out.WriteLine("After 9 years, the modified internal " +
"rate of return non the cows is " +
mirr.ToString("P");

Output

After 9 years, the modified internal rate of return
on the cows is 6.66 %

Example: Nominal Rate

In this example the nominal interest rate is computed given that the effective rate is 6.14% and that the
interest has been compounded quarterly.

using System;
using Imsl.Finance;

class nominalEx1
{
public static void Main(String[] args)
{
    double effectiveRate = .0614;
    int nper = 4;

    double nominalRate = Finance.Nominal(effectiveRate, nper);
    Console.Out.WriteLine("The nominal rate of the effective rate," +
    "6.14%, compounded quarterly is " +
    nominalRate.ToString("P");
}

Output

The nominal rate of the effective rate,6.14%, compounded quarterly is 6.00 %

Example: Number of Periods for an Investment

Someone obtains a $20,000 loan at 7.25% to buy a car. They want to make $350 a month payments.
Here, the number of payments necessary to pay off the loan is computed.

using System;
using Imsl.Finance;

public class nperEx1
{
    public static void Main(String[] args)
    {
        double rate = 0.0725 / 12;
        double pmt = -350.0;
        double pv = 20000;
        double fv = 0.0;
        double nperiods;
        nperiods = Finance.Nper(rate, pmt, pv, fv,
                                Finance.Period.AtBeginning);
        Console.Out.WriteLine("Number of payment periods = " + nperiods);
    }
}

Output
Number of payment periods = 69.7805113662826

Example: Net Present Value of an Investment
A lady wins a $10 million lottery. The money is to be paid out at the end of each year in $500,000 payments for 20 years. The current treasury bill rate of 6% is used as the discount rate. Here, the net present value of her prize is computed.

using System;
using Imsl.Finance;

public class npvEx1
{
    public static void Main(String[] args)
    {
        double rate = 0.06;
        double[] value_Renamed = new double[20];
        for (int i = 0; i < 20; i++)
            value_Renamed[i] = 500000.0;
        double npv = Finance.Npv(rate, value_Renamed);
        Console.Out.WriteLine("The net present value of the $10 " +
                               "million prize is " + npv.ToString("C"));
    }
}

Output
The net present value of the $10 million prize is $5,734,960.61
Example: Periodic Payments

The payment due each year on a 25 year, $100,000 loan is calculated. The loan is at 8%.

```csharp
using System;
using Imsl.Finance;

public class pmtEx1
{
    public static void Main(String[] args)
    {
        double rate = .08;
        int nper = 25;
        double pv = 100000.00;
        double fv = 0.0;

        double pmt = Finance.PeriodicPayment(rate, nper, pv, fv,
                                              Finance.Period.AtEnd);
        Console.Out.WriteLine("The payment due each year on the "$100,000 loan is " + pmt.ToString("C");
    }
}

Output

The payment due each year on the $100,000 loan is ($9,367.88)

Example: Principal Payments

The payment on the principal the first year on a 25 year, $100,000 loan is calculated. The loan is at 8%.

```csharp
using System;
using Imsl.Finance;

public class ppmtEx1
{
    public static void Main(String[] args)
    {
        double rate = .08;
        int per = 1;
        int nper = 25;
        double pv = 100000.00;
        double fv = 0.0;

        double ppmt = Finance.Ppmt(rate, per, nper, pv, fv,
                                     Finance.Period.AtEnd);
        Console.Out.WriteLine("The payment on the principal the first year of the $100,000 loan is " + ppmt.ToString("C");
    }
}
```
Output
The payment on the principal the first year of the $100,000 loan is ($1,367.88)

Example: Present Value of an Investment
A lady wins a $10 million lottery. The money is to be paid out at the end of each year in $500,000 payments for 20 years. The current treasury bill rate of 6% is used as the discount rate. Here, the present value of her prize is computed.

```csharp
using System;
using Imsl.Finance;

public class pvEx1
{
    public static void Main(String[] args)
    {
        double rate = 0.06;
        double pmt = 500000.0;
        double fv = 0.0;
        int nper = 20;

        double pv = Finance.Pv(rate, nper, pmt, fv,
                Finance.Period.AtEnd);

        Console.Out.WriteLine("The present value of the $10 million " +
                "prize is " + pv.ToString("C"));
    }
}
```

Output
The present value of the $10 million prize is ($5,734,960.61)

Example: Interest Rate
Someone obtains a $20,000 loan to buy a car. They make $350 a month payments for 70 months. Here, the interest rate of the loan is computed.

```csharp
using System;
using Imsl.Finance;

public class rateEx1
{
    public static void Main(String[] args)
    {
        int nper = 70;
        double pmt = -350.0;
        double pv = 20000;
        double fv = 0.0;

        double rate = 12.0 * Finance.Rate(nper, pmt, pv, fv,
                Finance.Period.AtEnd);
    }
}
```
Output
The computed interest rate on the loan is 7.35 %

Example: Depreciation - Straight Line Method
The straight line depreciation for one period of an asset with a life of 24 months, an initial cost of $2500 and a salvage value of $500 is computed.

```
using System;
using Imsl.Finance;

public class slnEx1
{
    public static void Main(String[] args)
    {
        double cost = 2500;
        double salvage = 500;
        int life = 24;

        double sln = Finance.Sln(cost, salvage, life);
        Console.Out.WriteLine("The straight line depreciation of the " +
        "asset for one period is " +
        sln.ToString("C"));
    }
}
```

Output
The straight line depreciation of the asset for one period is $83.33

Example: Depreciation - Sum-of-years’ Digits
The sum-of-years’ digits depreciation for the 14th year of an asset with a life of 15 years, an initial cost of $25000 and a salvage value of $5000 is computed.

```
using System;
using Imsl.Finance;

public class sydEx1
{
    public static void Main(String[] args)
    {
        double cost = 25000;
        double salvage = 5000;

```
Example: Depreciation - Variable Declining Balance

The depreciation between the 10th and 15th year of an asset with a life of 15 years, an initial cost of $25000 and a salvage value of $5000 is computed. The variable-declining balance method is used.

```csharp
using System;
using Imsl.Finance;

class vdbEx1
{
    static void Main(string[] args)
    {
        double cost = 25000;
        double salvage = 5000;
        int life = 15;
        int start = 10;
        int end = 15;
        double factor = 2.0;
        bool no_sl = false;

        double vdb = Finance.Vdb(cost, salvage, life, start, end, factor, no_sl);
        Console.WriteLine("The depreciation allowance between the {0}th and {1}th year is " + vdb.ToString("C");
    }
}
```

Output

The depreciation allowance between the 10th and 15th year is $976.69

Example: Internal Rate of Return - Variable Schedule

A farmer buys 10 young cows and a bull for $4500. The first year he does not expect to sell any calves, he just expects to feed them. Thereafter, he expects to be able to sell calves to offset the cost of feed. He expects them to be productive for 9 years, after which time he will liquidate the herd. The internal rate of return is computed after 9 years.
using System;
using Imsl.Finance;

public class xirrEx1
{
    public static void Main(String[] args)
    {
        double[] pmt = new double[]{-4500.0, -800.0, 800.0, 800.0, 600.0, 600.0, 800.0, 800.0, 700.0, 3000.0};
        System.DateTime[] dates = 
            new System.DateTime[] {DateTime.Parse("1/1/98"),
                DateTime.Parse("10/1/98"),
                DateTime.Parse("5/5/99"),
                DateTime.Parse("5/5/00"),
                DateTime.Parse("6/1/01"),
                DateTime.Parse("7/1/02"),
                DateTime.Parse("8/30/03"),
                DateTime.Parse("9/15/04"),
                DateTime.Parse("10/15/05"),
                DateTime.Parse("11/1/06");
        double xirr = Finance.Xirr(pmt, dates);
        Console.Out.WriteLine("After approximately 9 years, the " +
            "internal rate of return \n +
            "on the cows is " + xirr.ToString("P");
    }
}

Output
After approximately 9 years, the internal rate of return
on the cows is 7.69 %

Example: Present Value of a Schedule of Cash Flows
In this example, the present value of 3 payments, $1,000, $2,000, and $1,000, with an interest rate of 5%

using System;
using Imsl.Finance;

public class xnpvEx1
{
    public static void Main(String[] args)
    {
        double rate = 0.05;
        double[] value_Renamed = new double[]{1000.0, 2000.0, 1000.0};
        System.DateTime[] dates = 
            new System.DateTime[] {DateTime.Parse("1/3/1997"),
                DateTime.Parse("1/3/1999"),
                DateTime.Parse("1/3/2000");
    }
}
double pv = Finance.Xnpv(rate, value_Renamed, dates);
Console.Out.WriteLine("The present value of the schedule of " +
"cash flows is " + pv.ToString("C"));
}

Output
The present value of the schedule of cash flows is $3,677.90

---

**Finance.Period Enumeration**

public enumeration Imsl.Finance.Finance.Period
Used to indicate that payment is made at the beginning or end of each period.

**Fields**

AtBeginning
public Imsl.Finance.Finance.Period AtBeginning

Description
Indicates payment is made at the beginning of each period.

AtEnd
public Imsl.Finance.Finance.Period AtEnd

Description
Indicates payment is made at the end of each period.

---

**Bond Class**

public class Imsl.Finance.Bond
Collection of bond functions.
Definitions

*rate* is an annualized rate of return based on the par value of the bills.

*yield* is an annualized rate based on the purchase price and reflects the actual yield to maturity.

*coupons* are interest payments on a bond.

*redemption* is the amount a bond pays at maturity.

*frequency* is the number of times a year that a bond makes interest payments.

*basis* is the method used to calculate dates. For example, sometimes computations are done assuming 360 days in a year.

*issue* is the day a bond is first sold.

*settlement* is the day a purchaser acquires a bond.

*maturity* is the day a bond’s principal is repaid.

Discount Bonds

Discount bonds, also called *zero-coupon* bonds, do not pay interest during the life of the security, instead they sell at a discount to their value at maturity. The discount bond methods all have *settlement, maturity, basis* and *redemption* as arguments. In the following list these common arguments are omitted.

- price = Pricedisc(rate) (p. 1266)
- price = Priceyield(yield) (p. 1268)
- price = Pricemat(issue, rate, yield) (p. 1267)
- rate = Disc(price) (p. 1263)
- yield = Yielddisc(price) (p. 1272)

A related method is *Accrintm* (p. 1258), which returns the interest that has accumulated on the discount bond.

Treasury Bills

US Treasury bills are a special case of discount bonds. The *basis* is fixed for treasury bills and the redemption value is assumed to be $100. So these functions have only *settlement* and *maturity* as common arguments.

- price = Tbillprice(rate) (p. 1270)
- yield = Tbillyield(Price) (p. 1270)
- yield = Tbilleq(rate) (p. 1269)
**Interest Paying Bonds**

Most bonds pay interest periodically. The interest paying bond methods all have *settlement*, *maturity*, *basis* and *frequency* as arguments. Again supressing the common arguments,

- price = Price(rate, yield, redemption) (p. 1266)
- yield = Yield(rate, Price, redemption) (p. 1271)
- redemption = Received(Price, rate) (p. 1268)

A related method is Accrint (p. 1257), which returns the interest that has accumulated at settlement from the previous coupon date.

**Coupon days**

In this diagram, the settlement date is shown as a hollow circle and the adjacent coupon dates are shown as filled circles.
• Couppcd (p. 1263) is the coupon date immediately prior to the settlement date.
• Coupncd (p. 1262) is the coupon date immediately after the settlement date.
• Coupdays (p. 1260) is the number of days from the immediately prior coupon date to the settlement date.
• Coupdaysnc (p. 1261) is the number of days from the settlement date to the next Coupon date.
• Coupdays (p. 1261) is the number of days between these two coupon dates.

A related method is Coupnum (p. 1262), which returns the number of coupons payable between settlement and maturity.

Another related method is Yearfrac (p. 1271), which returns the fraction of the year between two days.

**Duration**

Duration is used to measure the sensitivity of a bond to changes in interest rates. Convexity is a measure of the sensitivity of duration.

• Duration (p. 1264)
• DayCountBasis modified duration (p. 1265)
• Convexity (p. 1259)

**Methods**

**Accrint**

static public double Accrint(System.DateTime issue, System.DateTime firstCoupon, System.DateTime settlement, double rate, double par, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis basis)

**Description**

Returns the interest which has accrued on a security that pays interest periodically.

**Parameters**

issue – The DateTime issue date of the security.
firstCoupon – The DateTime date of the security’s first interest date.
settlement – The DateTime settlement date of the security.
rate – A double which specifies the security’s annual coupon rate.
par – A double which specifies the security’s par value.
frequency – A int which specifies the number of coupon payments per year (1 for annual, 2 for semiannual, 4 for quarterly).
basis – A DayCountBasis object which contains the type of day count basis to use.
Returns
A double which specifies the accrued interest.

Remarks
In the equation below, $A_i$ represents the number of days which have accrued for the $i$th quasi-coupon period within the odd Frequency. (The quasi-coupon periods are periods obtained by extending the series of equal payment periods to before or after the actual payment periods.) $NC$ represents the number of quasi-coupon periods within the odd period, rounded to the next highest integer. (The odd period is a period between payments that differs from the usual equally spaced periods at which payments are made.) $NL_i$ represents the length of the normal $i$th quasi-coupon period within the odd Frequency. $NL_i$ is expressed in days. Accrint solves the following:

$$ par \left( \frac{rate}{frequency} \sum_{i=1}^{NC} \frac{A_i}{NL_i} \right) $$

### Accrintm

static public double Accrintm(System.DateTime issue, System.DateTime maturity, double rate, double par, Imsl.Finance.DayCountBasis basis)

Description
Returns the interest which has accrued on a security that pays interest at maturity.

Parameters
- `issue` – A `DateTime` issue date of the security.
- `maturity` – The `DateTime` date of the security’s maturity.
- `rate` – A double which specifies the security’s annual coupon rate.
- `par` – A double which specifies the security’s par value.
- `basis` – A `DayCountBasis` object which contains the type of day count basis to use.

Returns
A double which specifies the accrued interest.

Remarks

$$ = par \times rate \times \frac{A}{D} $$

In the above equation, $A$ represents the number of days starting at issue date to maturity date and $D$ represents the annual basis.

### Amordegrc

static public double Amordegrc(double cost, System.DateTime issue, System.DateTime firstPeriod, double salvage, int period, double rate, Imsl.Finance.DayCountBasis basis)
Description
Returns the depreciation for each accounting Frequency.

Parameters
- cost – A double which specifies the cost of the asset.
- issue – The DateTime issue date of the asset.
- firstPeriod – The DateTime date of the end of the first period.
- salvage – A double which specifies the asset’s salvage value at the end of the life of the asset.
- period – A int which specifies the period.
- rate – A double which specifies the rate of depreciation.
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the depreciation.

Remarks
This function is similar to Amorlinc. However, in this function a depreciation coefficient based on the asset life is applied during the evaluation of the function.

Amorlinc
static public double Amorlinc(double cost, System.DateTime issue, System.DateTime firstPeriod, double salvage, int period, double rate, Imsl.Finance.DayCountBasis basis)

Description
Returns the depreciation for each accounting Frequency.

Parameters
- cost – A double which specifies the cost of the asset.
- issue – The DateTime issue date of the asset.
- firstPeriod – The DateTime date of the end of the first period.
- salvage – A double which specifies the asset’s salvage value at the end of the life of the asset.
- period – A int which specifies the period.
- rate – A double which specifies the rate of depreciation.
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the depreciation.

Remarks
This function is similar to Amordegrc, except that Amordegrc has a depreciation coefficient that is applied during the evaluation that is based on the asset life.

Convexity
static public double Convexity(System.DateTime settlement, System.DateTime maturity, double coupon, double yield, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis basis)
Description
Returns the convexity for a security.

Parameters
- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- coupon – A double which specifies the security’s annual coupon rate.
- yield – A double which specifies the security’s annual yield.
- frequency – A int which specifies the number of coupon payments per year (1 for annual, 2 for semiannual, 4 for quarterly).
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the convexity for a security.

Remarks
Convexity is the sensitivity of the duration of a security to changes in yield. It is computed using the following:

\[
\frac{1}{(q \times \text{frequency})} \left\{ \sum_{t=1}^{n} t (t+1) \left( \frac{\text{coupon}}{\text{frequency}} q^{-t} + n (n+1) q^{-n} \right) \right\} - \left( \sum_{t=1}^{n} \frac{\text{coupon}}{\text{frequency}} q^{-t} + q^{-n} \right)
\]

where \( n \) is calculated from Coupnum, and \( q = 1 + \frac{\text{yield}}{\text{frequency}} \).

Coupdaybs
static public int Coupdaybs(System.DateTime settlement, System.DateTime maturity, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis basis)

Description
Returns the number of days starting with the beginning of the coupon period and ending with the settlement date.

Parameters
- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- frequency – A int which specifies the number of coupon payments per year.
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A int which specifies the number of days from the beginning of the coupon period to the settlement date.
Remarks
For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

**Coupdays**

static public double Coupdays(System.DateTime settlement, System.DateTime maturity, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis basis)

**Description**

Returns the number of days in the coupon period containing the settlement date.

**Parameters**

- **settlement** – The DateTime settlement date of the security.
- **maturity** – The DateTime maturity date of the security.
- **frequency** – A int which specifies the number of coupon payments per year.
- **basis** – A DayCountBasis object which contains the type of day count basis to use.

**Returns**

A int which specifies the number of days in the coupon period that contains the settlement date.

Remarks
For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

**Coupdaysnc**

static public int Coupdaysnc(System.DateTime settlement, System.DateTime maturity, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis basis)

**Description**

Returns the number of days starting with the settlement date and ending with the next coupon date.

**Parameters**

- **settlement** – The DateTime settlement date of the security.
- **maturity** – The DateTime maturity date of the security.
- **frequency** – A int which specifies the number of coupon payments per year.
- **basis** – A DayCountBasis object which contains the type of day count basis to use.

**Returns**

A int which specifies the number of days from the settlement date to the next coupon date.
Remarks
For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

**Coupncd**
```
static public System.DateTime Coupncd(System.DateTime settlement,
System.DateTime maturity, Imsl.Finance.Bond.Frequency frequency,
Imsl.Finance.DayCountBasis basis)
```

Description
Returns the first coupon date which follows the settlement date.

Parameters
- `settlement` – The `DateTime` settlement date of the security.
- `maturity` – The `DateTime` maturity date of the security.
- `frequency` – A `int` which specifies the number of coupon payments per year.
- `basis` – A `DayCountBasis` object which contains the type of day count basis to use.

Returns
A `int` which specifies the next coupon date after the settlement date.

Remarks
For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

**Coupnum**
```
static public int Coupnum(System.DateTime settlement, System.DateTime maturity,
```

Description
Returns the number of coupons payable between the settlement date and the maturity date.

Parameters
- `settlement` – The `DateTime` settlement date of the security.
- `maturity` – The `DateTime` maturity date of the security.
- `frequency` – A `int` which specifies the number of coupon payments per year.
- `basis` – A `DayCountBasis` object which contains the type of day count basis to use.

Returns
A `int` which specifies the number of coupons payable between the settlement date and maturity date.
Remarks
For a good discussion on day count basis, see SIA Standard Securities Calculation Methods 1993, vol. 1, pages 17-35.

Couppcd
static public System.DateTime Couppcd(System.DateTime settlement,
 System.DateTime maturity, Imsl.Finance.Bond.Frequency frequency,
 Imsl.Finance.DayCountBasis basis)

Description
Returns the coupon date which immediately precedes the settlement date.

Parameters
settlement – The DateTime settlement date of the security.
maturity – The DateTime maturity date of the security.
frequency – A int which specifies the number of coupon payments per year.
basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A int which specifies the previous coupon date before the settlement date.

Remarks
For a good discussion on day count basis, see SIA Standard Securities Calculation Methods 1993, vol. 1, pages 17-35.

Disc
static public double Disc(System.DateTime settlement, System.DateTime maturity,
 double price, double redemption, Imsl.Finance.DayCountBasis basis)

Description
Returns the implied interest rate of a discount bond.

Parameters
settlement – The DateTime settlement date of the security.
maturity – The DateTime maturity date of the security.
price – A double which specifies the security’s price per $100 face value.
redemption – A double which the security’s redemption value per $100 face value.
basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the discount rate for a security.
Remarks
The discount rate is the interest rate implied when a security is sold for less than its value at maturity in lieu of interest payments. It is computed using the following:

\[
\frac{\text{redemption} - \text{price}}{\text{price}} \times \frac{B}{DSM}
\]

In the equation above, \(B\) represents the number of days in a year based on the annual basis and \(DSM\) represents the number of days starting with the settlement date and ending with the maturity date.

Duration
static public double Duration(System.DateTime settlement, System.DateTime maturity, double coupon, double yield, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis basis)

Description
Returns the Macauley’s duration of a security where the security has periodic interest payments.

Parameters
- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- coupon – A double which specifies the security’s annual coupon rate.
- yield – A double which specifies the security’s annual yield.
- frequency – A int which specifies the number of coupon payments per year (1 for annual, 2 for semiannual, 4 for quarterly).
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the annual duration of a security with periodic interest payments.

Remarks
The Macauley’s duration is the weighted-average time to the payments, where the weights are the present value of the payments. It is computed using the following:

\[
\left(\frac{(N-1+E) \times 100}{(1+\frac{\text{yield}}{freq})^{(N-1+E)}}\right) + \sum_{k=1}^{N} \left(\frac{100 \times \text{coupon}}{freq \times \left(1+\frac{\text{yield}}{freq}\right)^{(k-1+E)}}\right) \times \left(1+\frac{\text{DSC}}{E}\right)\right) \times \frac{1}{freq}
\]

In the equation above, \(DSC\) represents the number of days starting with the settlement date and ending with the next coupon date. \(E\) represents the number of days within the coupon Frequency. \(N\) represents the number of coupons payable from the settlement date to the maturity date. \(freq\) represents the frequency of the coupon payments annually.
Intrate

static public double Intrate(System.DateTime settlement, System.DateTime maturity, double investment, double redemption, Imsl.Finance.DayCountBasis basis)

Description
Returns the interest rate of a fully invested security.

Parameters
- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- investment – A double which specifies the amount invested.
- redemption – A double which specifies the amount to be received at maturity.
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the interest rate for a fully invested security.

Remarks
It is computed using the following:

\[
\frac{\text{redemption} - \text{investment}}{\text{investment}} \times \frac{B}{DSM}
\]

In the equation above, \( B \) represents the number of days in a year based on the annual basis, and \( DSM \) represents the number of days in the period starting with the settlement date and ending with the maturity date.

Mduration

static public double Mduration(System.DateTime settlement, System.DateTime maturity, double coupon, double yield, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis basis)

Description
Returns the modified Macauley duration for a security with an assumed par value of $100.

Parameters
- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- coupon – A double which specifies the security’s annual coupon rate.
- yield – A double which specifies the security’s annual yield.
- frequency – A int which specifies the number of coupon payments per year (1 for annual, 2 for semiannual, 4 for quarterly).
- basis – A DayCountBasis object which contains the type of day count basis to use.
Returns
A double which specifies the modified Macauley duration for a security with an assumed par value of $100.

Remarks
It is computed using the following:

\[
\frac{duration}{1 + \frac{yield}{frequency}}
\]

where \(duration\) is calculated from \(Mduration\).

Price
static public double Price(System.DateTime settlement, System.DateTime maturity, double rate, double yield, double redemption, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis basis)

Description
Returns the price, per $100 face value, of a security that pays periodic interest.

Parameters
- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- rate – A double which specifies the security’s annual coupon rate.
- yield – A double which specifies the security’s annual yield.
- redemption – A double which specifies the security’s redemption value per $100 face value.
- frequency – A int which specifies the number of coupon payments per year (1 for annual, 2 for semiannual, 4 for quarterly).
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the price per $100 face value of a security that pays periodic interest.

Remarks
It is computed using the following:

\[
\frac{redemption}{\left(1 + \frac{yield}{frequency}\right)^{(N-1 + \frac{DSC}{E})}} + \sum_{k=1}^{N} \left(1 + \frac{100 \times \frac{rate}{frequency}}{\left(1 + \frac{yield}{frequency}\right)^{(k-1 + \frac{DSC}{E})}} - \left(100 \times \frac{rate}{frequency} \times \frac{A}{E}\right)\right)
\]

In the above equation, \(DSC\) represents the number of days in the period starting with the settlement date and ending with the next coupon date. \(E\) represents the number of days within the coupon Frequency. \(N\) represents the number of coupons payable in the timeframe from the settlement date to the redemption date. \(A\) represents the number of days in the timeframe starting with the beginning of coupon period and ending with the settlement date.

Pricedisc
static public double Pricedisc(System.DateTime settlement, System.DateTime maturity, double rate, double redemption, Imsl.Finance.DayCountBasis basis)
Description
Returns the price of a discount bond given the discount rate.

Parameters
- **settlement** – The DateTime settlement date of the security.
- **maturity** – The DateTime maturity date of the security.
- **rate** – A double which specifies the security's discount rate.
- **redemption** – A double which specifies the security's redemption value per $100 face value.
- **basis** – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the price per $100 face value of a discounted security.

Remarks
It is computed using the following:

\[
\text{redemption} - \text{rate} \times \text{redemption} \times \frac{DSM}{B}
\]

In the equation above, \(DSM\) represents the number of days starting at the settlement date and ending with the maturity date. \(B\) represents the number of days in a year based on the annual basis.

Pricemat
static public double Pricemat(System.DateTime settlement, System.DateTime maturity, System.DateTime issue, double rate, double yield, Imsl.Finance.DayCountBasis basis)

Description
Returns the price, per $100 face value, of a discount bond.

Parameters
- **settlement** – The DateTime settlement date of the security.
- **maturity** – The DateTime maturity date of the security.
- **issue** – The DateTime issue date of the security.
- **rate** – A double which specifies the security's interest rate at issue date.
- **yield** – A double which specifies the security's annual yield.
- **basis** – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the price per $100 face value of a security that pays interest at maturity.
Remarks
It is computed using the following:

\[ \frac{100 + \left( \frac{\text{DIM}}{B} \times \text{rate} \times 100 \right)}{1 + \left( \frac{\text{DSM}}{B} \times \text{yield} \right)} - \frac{A}{B} \times \text{rate} \times 100 \]

In the equation above, \( B \) represents the number of days in a year based on the annual basis. \( DSM \) represents the number of days in the period starting with the settlement date and ending with the maturity date. \( DIM \) represents the number of days in the period starting with the issue date and ending with the maturity date. \( A \) represents the number of days in the period starting with the issue date and ending with the settlement date.

Priceyield
static public double Priceyield(System.DateTime settlement, System.DateTime maturity, double yield, double redemption, Imsl.Finance.DayCountBasis basis)

Description
Returns the price of a discount bond given the yield.

Parameters
- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- yield – A double which specifies the security’s yield.
- redemption – A double which specifies the security’s redemption value per $100 face value.
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the price per $100 face value of a discounted security.

Remarks
It is computed using the following:

\[ \frac{\text{redemption}}{1 + \left( \frac{\text{DSM}}{B} \right) \times \text{yield}} \]

In the equation above, \( DSM \) represents the number of days starting at the settlement date and ending with the maturity date. \( B \) represents the number of days in a year based on the annual basis.

Received
static public double Received(System.DateTime settlement, System.DateTime maturity, double investment, double rate, Imsl.Finance.DayCountBasis basis)

Description
Returns the amount one receives when a fully invested security reaches the maturity date.
Parameters

- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- investment – A double which specifies the amount invested in the security.
- rate – A double which specifies the security’s rate at issue date.
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns

A double which specifies the amount received at maturity for a fully invested security.

Remarks

It is computed using the following:

\[
\frac{investment}{1 - (rate \times \frac{DIM}{B})}
\]

In the equation above, \( B \) represents the number of days in a year based on the annual basis, and \( DIM \) represents the number of days in the period starting with the issue date and ending with the maturity date.

Tbilleq

static public double Tbilleq(System.DateTime settlement, System.DateTime maturity, double rate)

Description

Returns the bond-equivalent yield of a Treasury bill.

Parameters

- settlement – The DateTime settlement date of the Treasury bill.
- maturity – The DateTime maturity date of the Treasury bill. The maturity cannot be more than a year after the settlement.
- rate – A double which specifies the Treasury bill’s discount rate at issue date. The discount rate is an annualized rate of return based on the par value of the bills. The discount rate is calculated on a 360-day basis (twelve 30-day months).

Returns

A double which specifies the bond-equivalent yield for the Treasury bill. This is an annualized rate based on the purchase price of the bills and reflects the actual yield to maturity.

Remarks

It is computed using the following: If \( DSM \leq 182 \)

\[
\frac{365 \times rate}{360 - rate \times DSM}
\]

otherwise,

\[
-\frac{DSM}{365} + \sqrt{\left(\frac{DSM}{365}\right)^2 - 2 \times \frac{DSM}{365} - 1 \times \frac{rate \times DSM}{rate \times DSM - 360}}
\]

\[
\frac{DSM}{365} - 0.5
\]

In the above equation, \( DSM \) represents the number of days starting at settlement date to maturity date.
**Tbillprice**

static public double Tbillprice(System.DateTime settlement, System.DateTime maturity, double rate)

**Description**

Returns the price, per $100 face value, of a Treasury bill.

**Parameters**

- settlement – The DateTime settlement date of the Treasury bill.
- maturity – The DateTime maturity date of the Treasury bill. The maturity cannot be more than a year after the settlement.
- rate – A double which specifies the Treasury bill’s discount rate at issue date. The discount rate is an annualized rate of return based on the par value of the bills. The discount rate is calculated on a 360-day basis (twelve 30-day months).

**Returns**

A double which specifies the price per $100 face value for the Treasury bill.

**Remarks**

It is computed using the following:

$$\frac{100 \left( 1 - \frac{rate \times DSM}{360} \right)}{1 - \frac{rate \times DSM}{360}}$$

In the equation above, DSM represents the number of days in the period starting with the settlement date and ending with the maturity date (any maturity date that is more than one calendar year after the settlement date is excluded).

**Tbillyield**

static public double Tbillyield(System.DateTime settlement, System.DateTime maturity, double price)

**Description**

Returns the yield of a Treasury bill.

**Parameters**

- settlement – The DateTime settlement date of the Treasury bill.
- maturity – The DateTime maturity date of the Treasury bill. The maturity cannot be more than a year after the settlement.
- price – A double which specifies the Treasury bill’s price per $100 face value.

**Returns**

A double which specifies the yield for the Treasury bill. This is an annualized rate based on the purchase price of the bills and reflects the actual yield to maturity.
Remarks

It is computed using the following:

\[
\frac{100 - price}{price} \times \frac{360}{DSM}
\]

In the equation above, DSM represents the number of days in the period starting with the settlement date and ending with the maturity date (any maturity date that is more than one calendar year after the settlement date is excluded).

Yearfrac

static public double Yearfrac(System.DateTime startDate, System.DateTime endDate, Imsl.Finance.DayCountBasis basis)

Description

Returns the fraction of a year represented by the number of whole days between two dates.

Parameters

startDate – The DateTime start date of the security.

endDate – The DateTime end date of the security.

basis – A DayCountBasis object which contains the type of day count basis to use.

Returns

A double which specifies the annual yield of a security that pays interest at maturity.

Remarks

It is computed using the following:

\[
A/D
\]

where \(A\) equals the number of days from start to end, \(D\) equals annual basis.

Yield

static public double Yield(System.DateTime settlement, System.DateTime maturity, double rate, double price, double redemption, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis basis)

Description

Returns the yield of a security that pays periodic interest.

Parameters

settlement – The DateTime settlement date of the security.

maturity – The DateTime maturity date of the security.

rate – A double which specifies the security’s annual coupon rate.

price – A double which specifies the security’s price per $100 face value.

redemption – A double which specifies the security’s redemption value per $100 face value.

frequency – A int which specifies the number of coupon payments per year (1 for annual, 2 for semiannual, 4 for quarterly).

basis – A DayCountBasis object which contains the type of day count basis to use.
Returns
A double which specifies the yield of a security that pays periodic interest.

Remarks
If there is one coupon period use the following:

\[
\left( \frac{\text{redemption}}{100} + \frac{\text{rate}}{\text{frequency}} \right) - \left[ \frac{\text{price}}{100} + \left( A \times \frac{\text{rate}}{\text{frequency}} \right) \right] \times \frac{\text{frequency} \times E}{DSR}
\]

In the equation above, \( DSR \) represents the number of days in the period starting with the settlement date and ending with the redemption date. \( E \) represents the number of days within the coupon Frequency. \( A \) represents the number of days in the period starting with the beginning of coupon period and ending with the settlement date.

If there is more than one coupon period use the following:

\[
\text{price} - \frac{\text{redemption}}{\left( 1 + \frac{\text{yield}}{\text{frequency}} \right)^{N - 1}} - \left( \sum_{k=1}^{N} \frac{100 \times \frac{\text{rate}}{\text{frequency}}}{\left( 1 + \frac{\text{yield}}{\text{frequency}} \right)^{k-1}} \right) + 100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{A}{E} = 0
\]

In the equation above, \( DSC \) represents the number of days in the period from the settlement to the next coupon date. \( E \) represents the number of days within the coupon Frequency. \( N \) represents the number of coupons payable in the period starting with the settlement date and ending with the redemption date. \( A \) represents the number of days in the period starting with the beginning of the coupon period and ending with the settlement date.

Yielddisc

static public double Yielddisc(System.DateTime settlement, System.DateTime maturity, double price, double redemption, Imsl.Finance.DayCountBasis basis)

Description
Returns the annual yield of a discount bond.

Parameters
- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- price – A double which specifies the security’s price per $100 face value.
- redemption – A double which specifies the security’s redemption value per $100 face value.
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the annual yield for a discounted security.
Remarks
It is computed using the following:

\[
\frac{\text{redemption} - \text{price}}{\text{price}} \times \frac{B}{DSM}
\]

In the equation above, \( B \) represents the number of days in a year based on the annual basis, and \( DSM \) represents the number of days starting with the settlement date and ending with the maturity date.

Yieldmat

\[
\text{static public double Yieldmat(System.DateTime settlement, System.DateTime maturity, System.DateTime issue, double rate, double price, Imsl.Finance.DayCountBasis basis)}
\]

Description
Returns the annual yield of a security that pays interest at maturity.

Parameters
- settlement – The DateTime settlement date of the security.
- maturity – The DateTime maturity date of the security.
- issue – The DateTime issue date of the security.
- rate – A double which specifies the security’s interest rate at date of issue.
- price – A double which specifies the security’s price per $100 face value.
- basis – A DayCountBasis object which contains the type of day count basis to use.

Returns
A double which specifies the annual yield of a security that pays interest at maturity.

Remarks
It is computed using the following:

\[
\frac{\left[1 + \left(\frac{DIM}{B} \times \text{rate}\right)\right] - \left[\frac{\text{price}}{100} + \left(\frac{A}{B} \times \text{rate}\right)\right]}{\left[\frac{\text{price}}{100} + \left(\frac{A}{B} \times \text{rate}\right)\right]} \times \frac{B}{DSM}
\]

In the equation above, \( DIM \) represents the number of days in the period starting with the issue date and ending with the maturity date. \( DSM \) represents the number of days in the period starting with the settlement date and ending with the maturity date. \( A \) represents the number of days in the period starting with the issue date and ending with the settlement date. \( B \) represents the number of days in a year based on the annual basis.

Example: Accrued Interest - Periodic Payments
In this example, the accrued interest is calculated for a bond which pays interest semiannually. The day count basis used is 30/360.
using System;
using Imsl.Finance;

public class accrintEx1
{
    public static void Main(String[] args)
    {
        DateTime issue = DateTime.Parse("10/1/91");
        DateTime firstCoupon = DateTime.Parse("3/31/92");
        DateTime settlement = DateTime.Parse("11/3/91");
        double rate = .06;
        double par = 1000.0;
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double accrint = Bond.Accrint(issue, firstCoupon, settlement,
                                      rate, par, freq, dcb);
        Console.Out.WriteLine("The accrued interest is "+ accrint);
    }
}

Output
The accrued interest is 5.33333333333333

Example: Accrued Interest - Payment at Maturity
In this example, the accrued interest is calculated for a bond which pays at maturity. The day count basis used is 30/360.

using System;
using Imsl.Finance;

public class accrintmEx1
{
    public static void Main(String[] args)
    {
        DateTime issue = DateTime.Parse("10/1/91");
        DateTime settlement = DateTime.Parse("11/3/91");
        double rate = .06;
        double par = 1000.0;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double accrintm = Bond.Accrintm(issue, settlement, rate, par, dcb);
        Console.Out.WriteLine("The accrued interest is "+ accrintm);
    }
}

Output
The accrued interest is 5.33333333333333

Example: Depreciation - French Accounting System
In this example, the depreciation for the second accounting period is calculated for an asset.
using System;
using Imsl.Finance;

public class amordegrcEx1
{
    public static void Main(String[] args)
    {
        double cost = 2400.0;
        DateTime issue = DateTime.Parse("11/1/92");
        DateTime firstPeriod = DateTime.Parse("11/30/93");
        double salvage = 300.0;
        int period = 2;
        double rate = .15;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double amordegrc = Bond.Amordegrc(cost, issue, firstPeriod,
                                          salvage, period, rate, dcb);
        Console.Out.WriteLine("The depreciation for the second accounting "
                              + "period is " + amordegrc);
    }
}

Output
The depreciation for the second accounting period is 334

Example: Depreciation - French Accounting System
In this example, the depreciation for the second accounting period is calculated for an asset.

using System;
using Imsl.Finance;

public class amorlincEx1
{
    public static void Main(String[] args)
    {
        double cost = 2400.0;
        DateTime issue = DateTime.Parse("11/1/92");
        DateTime firstPeriod = DateTime.Parse("11/30/93");
        double salvage = 300.0;
        int period = 2;
        double rate = .15;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double amorlinc = Bond.Amorlinc(cost, issue, firstPeriod,
                                         salvage, period, rate, dcb);
        Console.Out.WriteLine("The depreciation for the second accounting "
                              + "period is " + amorlinc);
    }
}

Output
The depreciation for the second accounting period is 360
Example: Convexity for a Security

The convexity of a 10 year bond which pays interest semiannually is returned in this example.

```csharp
using System;
using Imsl.Finance;

public class convexityEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/90");
        DateTime maturity = DateTime.Parse("7/1/00");
        double coupon = .075;
        double yield = .09;
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        double convexity = Bond.Convexity(settlement, maturity, coupon,
                                           yield, freq, dcb);
        Console.Out.WriteLine("The convexity of the bond with semiannual "
                              + "interest payments is " + convexity);
    }
}
```

Output

The convexity of the bond with semiannual interest payments is 59.4049912915856

Example: Days - Beginning of Period to Settlement Date

In this example, the settlement date is 11/11/86. The number of days from the beginning of the coupon period to the settlement date is returned.

```csharp
using System;
using Imsl.Finance;

public class coupdaybsEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("11/11/86");
        DateTime maturity = DateTime.Parse("3/1/99");
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        int coupdaybs = Bond.Coupdaybs(settlement, maturity, freq, dcb);
        Console.Out.WriteLine("The number of days from the beginning of the"
                              + " coupon period to the settlement date is "
                              + coupdaybs);
    }
}
```

Output

The number of days from the beginning of the
coupon period to the settlement date is 71

**Example: Days in the Settlement Date Period**

In this example, the settlement date is 11/11/86. The number of days in the coupon period containing this date is returned.

```csharp
using System;
using Imsl.Finance;

public class coupdaysEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("11/11/86");
        DateTime maturity = DateTime.Parse("3/1/99");
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        double coupdays = Bond.Coupdays(settlement, maturity, freq, dcb);
        Console.Out.WriteLine("The number of days in the coupon period that "
                                                  + "contains the settlement date is "+ coupdays);
    }
}
```

**Output**
The number of days in the coupon period that contains the settlement date is 182.5

**Example: Days - Settlement Date to Next Coupon Date**

In this example, the settlement date is 11/11/86. The number of days from this date to the next coupon date is returned.

```csharp
using System;
using Imsl.Finance;

public class coupdaysncEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("11/11/86");
        DateTime maturity = DateTime.Parse("3/1/99");
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        int coupdaysnc = Bond.Coupdaysnc(settlement, maturity, freq, dcb);
        Console.Out.WriteLine("The number of days from the settlement date "
                                                  + "to the next coupon date is " + coupdaysnc);
    }
}
```
Output
The number of days from the settlement date to the next coupon date is 110

Example: Next Coupon Date After the Settlement Date
In this example, the settlement date is 11/11/86. The previous coupon date before this date is returned.

using System;
using Imsl.Finance;

public class coupncdEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("11/11/86");
        DateTime maturity = DateTime.Parse("3/1/99");
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        DateTime coupncd = Bond.Coupncd(settlement, maturity, freq, dcb);
        Console.Out.WriteLine("The next coupon date after the " +
                              "settlement date is " + coupncd);
    }
}

Output
The next coupon date after the settlement date is 3/1/1987 12:00:00 AM

Example: Number of Payable Coupons
In this example, the settlement date is 11/11/86. The number of payable coupons between this date and the maturity date is returned.

using System;
using Imsl.Finance;

public class coupnumEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("11/11/86");
        DateTime maturity = DateTime.Parse("3/1/99");
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        int coupnum = Bond.Coupnum(settlement, maturity, freq, dcb);
        Console.Out.WriteLine("The number of coupons payable between" +
                              " the settlement date and the maturity" +
                              " date is " + coupnum);
    }
}
Output
The number of coupons payable between the settlement date and the maturity date is 25

Example: Previous Coupon Date Before the Settlement Date
In this example, the settlement date is 11/11/86. The previous coupon date before this date is returned.

```
using System;
using Imsl.Finance;

public class couppcdEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("11/11/86");
        DateTime maturity = DateTime.Parse("3/1/99");
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        DateTime couppcd = Bond.Couppcd(settlement, maturity, freq, dcb);
        Console.Out.WriteLine("The previous coupon date before the " +
                             "settlement date is " +
                             couppcd.ToLongDateString());
    }
}
```

Output
The previous coupon date before the settlement date is Monday, September 01, 1986

Example: Discount Rate for a Security
In this example, the discount rate for a security is returned.

```
using System;
using Imsl.Finance;

public class discEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("2/15/92");
        DateTime maturity = DateTime.Parse("6/10/92");
        double price = 97.975;
        double redemption = 100.0;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        double disc = Bond.Disc(settlement, maturity, price,
                                 redemption, dcb);
        Console.Out.WriteLine("The discount rate for the security is " +
                              disc);
    }
}
```
Output
The discount rate for the security is 0.0637176724137933

Example: Duration of a Security with Periodic Payments
The annual duration of a 10 year bond which pays interest semiannually is returned in this example.

```csharp
using System;
using Imsl.Finance;

class durationEx1
{
    public static void Main(string[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/95");
        double coupon = 0.075;
        double yield = 0.09;
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        double duration = Bond.Duration(settlement, maturity, coupon,
                                        yield, freq, dcb);
        Console.WriteLine("The annual duration of the bond with semiannual interest payments is "+duration);
    }
}
```

Output
The annual duration of the bond with semiannual interest payments is 7.04195337797215

Example: Interest Rate of a Fully Invested Security
The discount rate of a 10 year bond is returned in this example.

```csharp
using System;
using Imsl.Finance;

class intrateEx1
{
    public static void Main(string[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/95");
        double investment = 7000.0;
        double rate = Bond.Intrate(settlement, maturity, investment);
        Console.WriteLine("The discount rate of the bond is "+rate);
    }
}
```
double redemption = 10000.0;
DayCountBasis dcb = DayCountBasis.BasisActual365;
double intrate = Bond.Intrate(settlement, maturity, investment,
    redemption, dcb);
Console.Out.WriteLine("The interest rate of the bond is "+intrate);
}

Output
The interest rate of the bond is 0.0428336723517446

Example: Modified Macauley Duration of a Security with Periodic Payments

The modified Macauley duration of a 10 year bond which pays interest semiannually is returned in this example.

using System;
using Imsl.Finance;

public class mdurationEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/95");
        double coupon = .075;
        double yield = .09;
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        double mduration = Bond.Mduration(settlement, maturity, coupon,
            yield, freq, dcb);
        Console.Out.WriteLine("The modified Macauley duration "+
            "of the bond");
        Console.Out.WriteLine("with semiannual interest payments is "+
            mduration);
    }
}

Output
The modified Macauley duration of the bond with semiannual interest payments is 6.73871136648053

Example: Price of a Security

The price per $100 face value of a 10 year bond which pays interest semiannually is returned in this example.
using System;
using Imsl.Finance;

public class priceEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/95");
        double rate = .06;
        double yield = .07;
        double redemption = 105.0;
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double price = Bond.Price(settlement, maturity, rate, yield, redemption, freq, dcb);
        Console.Out.WriteLine("The price of the bond is "+price.ToString("C"));
    }
}

Output
The price of the bond is $95.41

Example: Price of a Discounted Security
The price per $100 face value of a discounted 1 year bond is returned in this example.

using System;
using Imsl.Finance;

public class pricediscEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/86");
        double rate = .05;
        double redemption = 100.0;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double pricedisc = Bond.Pricedisc(settlement, maturity, rate, redemption, dcb);
        Console.Out.WriteLine("The price of the discounted bond is "+pricedisc.ToString("C"));
    }
}

Output
The price of the discounted bond is $95.00
Example: Price of a Security that Pays at Maturity

The price per $100 face value of 1 year bond that pays interest at maturity is returned in this example.

```csharp
using System;
using Imsl.Finance;

class pricematEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("8/1/85");
        DateTime maturity = DateTime.Parse("7/1/86");
        DateTime issue = DateTime.Parse("7/1/85");
        double rate = .05;
        double yield = .05;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double pricemat = Bond.Pricemat(settlement, maturity, issue, rate, yield, dcb);
        Console.Out.WriteLine("The price of the bond is "+pricemat);
    }
}
```

Output

The price of the bond is 99.9817397078353

Price of a Discounted Security

The price of a discounted 1 year bond is returned in this example.

```csharp
using System;
using Imsl.Finance;

class priceyieldEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/95");
        double yield = 0.010055244588347783;
        double redemption = 105.0;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double priceyield = Bond.Priceyield(settlement, maturity, yield, redemption, dcb);
        Console.Out.WriteLine("The price of the discounted bond is "+priceyield);
    }
}
```
Output
The price of the discounted bond is 95.40663

Example: Amount Received at Maturity for a Fully Invested Security
The amount to be received at maturity for a 10 year bond is returned in this example.

```csharp
using System;
using Imsl.Finance;

public class receivedEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/95");
        double investment = 7000.0;
        double discount = .06;
        DayCountBasis dcb = DayCountBasis.BasisActual365;
        double received = Bond.Received(settlement, maturity, investment, discount, dcb);
        Console.Out.WriteLine("The amount received at maturity for the bond is "+ received.ToString("C"));
    }
}
```

Output
The amount received at maturity for the bond is $17,514.40

Example: Bond-Equivalent Yield
The bond-equivalent yield for a 1 year Treasury bill is returned in this example.

```csharp
using System;
using Imsl.Finance;

public class tbilleqEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/86");
        double discount = .05;
        double tbilleq = Bond.Tbilleq(settlement, maturity, discount);
        Console.Out.WriteLine("The bond-equivalent yield for the T-bill is "+ tbilleq.ToString("P"));
    }
}
```
Output
The bond-equivalent yield for the T-bill is 5.27 %

Example: Treasury Bill Price
The price per $100 face value for a 1 year Treasury bill is returned in this example.

```java
using System;
using Imsl.Finance;

public class tbillpriceEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/86");
        double discount = .05;
        double tbillprice = Bond.Tbillprice(settlement, maturity, discount);
        Console.Out.WriteLine("The price per $100 face value for the " +
                           "T-bill is " + tbillprice.ToString("C");
    }
}
```

Output
The price per $100 face value for the T-bill is $94.93

Example: Treasury Bill Yield
The yield for a 1 year Treasury bill is returned in this example.

```java
using System;
using Imsl.Finance;

public class tbillyieldEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/86");
        double price = 94.93;
        double tbillyield = Bond.Tbillyield(settlement, maturity, price);
        Console.Out.WriteLine("The yield for the T-bill is " +
                           tbillyield.ToString("P");
    }
}
```

Output
The yield for the T-bill is 5.27 %
Example: Year Fraction
The year fraction of a 30/360 year starting 8/1/85 and ending 7/1/86 is returned in this example.

```csharp
using System;
using Imsl.Finance;

public class yearfracEx1
{
    public static void Main(String[] args)
    {
        DateTime start = DateTime.Parse("8/1/85");
        DateTime end = DateTime.Parse("7/1/86");
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double yearfrac = Bond.Yearfrac(start, end, dcb);
        Console.Out.WriteLine("The year fraction of the 30/360 period " + "is " + yearfrac);
    }
}
```

Output
The year fraction of the 30/360 period is 0.916666666666667

Example: Yield on a Security
The yield on a 10 year bond which pays interest semiannually is returned in this example.

```csharp
using System;
using Imsl.Finance;

public class yieldEx1
{
    public static void Main(String[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/95");
        double rate = .06;
        double price = 95.40663;
        double redemption = 105.0;
        Bond.Frequency freq = Bond.Frequency.SemiAnnual;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double yield = Bond.Yield(settlement, maturity, rate, price, redemption, freq, dcb);
        Console.Out.WriteLine("The yield of the bond is " + yield);
    }
}
```

Output
The yield of the bond is 0.0699999968284289
Example: Yield on a Discounted Security

The yield on a discounted 10 year bond is returned in this example.

```csharp
using System;
using Imsl.Finance;

class yielddiscEx1
{
    public static void Main(string[] args)
    {
        DateTime settlement = DateTime.Parse("7/1/85");
        DateTime maturity = DateTime.Parse("7/1/95");
        double price = 95.40663;
        double redemption = 105.0;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double yielddisc = Bond.Yielddisc(settlement, maturity, price, redemption, dcb);
        Console.WriteLine("The yield on the discounted bond is "+yielddisc);
    }
}
```

Output

The yield on the discounted bond is 0.0100552445883478

Example: Yield on a Security Which Pays at Maturity

The yield on a bond which pays at maturity is returned in this example.

```csharp
using System;
using Imsl.Finance;

class yieldmatEx1
{
    public static void Main(string[] args)
    {
        DateTime settlement = DateTime.Parse("8/1/85");
        DateTime maturity = DateTime.Parse("7/1/95");
        DateTime issue = DateTime.Parse("7/1/85");
        double rate = .06;
        double price = 95.40663;
        DayCountBasis dcb = DayCountBasis.BasisNASD;
        double yieldmat = Bond.Yieldmat(settlement, maturity, issue, rate, price, dcb);
        Console.WriteLine("The yield on a bond which pays at "+
maturity is "+yieldmat);
    }
}
```

Output

The yield on a bond which pays at maturity is 0.0673905127809195
Bond.Frequency Enumeration

public enumeration Imsl.Finance.Bond.Frequency

Frequency of the bond's coupon payments.

## Fields

**Annual**

public Imsl.Finance.Bond.Frequency Annual

**Description**

Indicates interest is paid once a year.

**BiMonthly**

public Imsl.Finance.Bond.Frequency BiMonthly

**Description**

Indicates interest is paid six times a year.

**Monthly**

public Imsl.Finance.Bond.Frequency Monthly

**Description**

Indicates interest is paid twelve times a year.

**Quarterly**

public Imsl.Finance.Bond.Frequency Quarterly

**Description**

Indicates interest is paid four times a year.

**SemiAnnual**

public Imsl.Finance.Bond.Frequency SemiAnnual

**Description**

Indicates interest is paid twice a year.
DayCountBasis Class

public class Imsl.Finance.DayCountBasis

The Day Count Basis.

Rules for computing the number of days between two dates or number of days in a year. For many securities, computations are based on rules other than on the actual calendar.

Fields

**Basis30e360**

public Imsl.Finance.DayCountBasis Basis30e360

**Description**

Computations based on the assumption of 30 days per month and 360 days per year.

See Also: Imsl.Finance.DayCountBasis.BasisPart30E360 (p. 1290)

**BasisActual360**

public Imsl.Finance.DayCountBasis BasisActual360

**Description**

Computations are based on the number of days in a month based on the actual calendar value and the number of days, but assuming 360 days per year.


**BasisActual365**

public Imsl.Finance.DayCountBasis BasisActual365

**Description**

Computations are based on the number of days in a month based on the actual calendar value and the number of days, but assuming 365 days per year.


**BasisActualActual**

public Imsl.Finance.DayCountBasis BasisActualActual
Description
Computations are based on the actual calendar.
See Also: Imsl.Finance.DayCountBasis.BasisPartActual (p. 1290)

BasisNASD
public Imsl.Finance.DayCountBasis BasisNASD

Description
Computations based on the assumption of 30 days per month and 360 days per year.
See Also: Imsl.Finance.DayCountBasis.BasisPartNASD (p. 1290)

BasisPart30E360
public Imsl.Finance.IBasisPart BasisPart30E360

Description
Computations based on the assumption of 30 days per month and 360 days per year. This computes the number of days between two dates differently than BasisPartNASD for months with other than 30 days.

BasisPart365
public Imsl.Finance.IBasisPart BasisPart365

Description
Computations based on the assumption of 365 days per year.

BasisPartActual
public Imsl.Finance.IBasisPart BasisPartActual

Description
Computations are based on the actual calendar.

BasisPartNASD
public Imsl.Finance.IBasisPart BasisPartNASD

Description
Computations based on the assumption of 30 days per month and 360 days per year.

Properties

MonthBasis
public Imsl.Finance.IBasisPart MonthBasis {get; }
**Property Value**
A IBasisPart object which represents the month Basis for this DayCountBasis.

**YearBasis**
public Imsl.Finance.IBasisPart YearBasis {get; }

**Description**
The (days in year) portion of the Day Count Basis.

**Property Value**
A IBasisPart object which represents the year Basis for this DayCountBasis.

---

**Constructor**

**DayCountBasis**
public DayCountBasis(Imsl.Finance.IBasisPart monthBasis, Imsl.Finance.IBasisPart yearBasis)

**Description**
Creates a new DayCountBasis.

**Parameters**
- monthBasis – A IBasisPart which specifies the month basis.
- yearBasis – A IBasisPart which specifies the year basis.

---

**IBasisPart Interface**

public interface Imsl.Finance.IBasisPart
Component of DayCountBasis.
The day count basis consists of a month basis and a yearly basis. Each of these components implements this interface.

---

**See Also**

Imsl.Finance.DayCountBasis (p. 1289)
Methods

DaysBetween
abstract public int DaysBetween(System.DateTime date1, System.DateTime date2)

Description
Returns the number of days from date1 to date2.

Parameters
date1 – A DateTime object containing the initial date.
date2 – A DateTime object containing the final date.

Returns
A int which specifies the number of days from date1 to date2.

DaysInPeriod
abstract public double DaysInPeriod(System.DateTime finalDate,
Imsl.Finance.Bond.Frequency frequency)

Description
Returns the number of days in a coupon period.

Parameters
finalDate – A DateTime object containing the final date of the coupon period.
frequency – The Frequency specifying the number of coupon periods per year. This is typically 1, 2 or 4.

Returns
A int containing the number of days in the coupon period.

GetDaysInYear
abstract public int GetDaysInYear(System.DateTime date)

Description
Returns the number of days in the year.

Parameter
date – A DateTime object containing the date.

Returns
A int which specifies the number of days in the year.
# Chapter 24: Chart2D

## Types

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractChartNode</td>
<td>1294</td>
</tr>
<tr>
<td>ChartNode</td>
<td>1310</td>
</tr>
<tr>
<td>Chart</td>
<td>1332</td>
</tr>
<tr>
<td>Background</td>
<td>1338</td>
</tr>
<tr>
<td>ChartTitle</td>
<td>1339</td>
</tr>
<tr>
<td>Grid</td>
<td>1340</td>
</tr>
<tr>
<td>Legend</td>
<td>1341</td>
</tr>
<tr>
<td>Annotation</td>
<td>1342</td>
</tr>
<tr>
<td>Axis</td>
<td>1346</td>
</tr>
<tr>
<td>AxisXY</td>
<td>1348</td>
</tr>
<tr>
<td>Axis1D</td>
<td>1353</td>
</tr>
<tr>
<td>AxisLabel</td>
<td>1358</td>
</tr>
<tr>
<td>AxisLine</td>
<td>1359</td>
</tr>
<tr>
<td>AxisTitle</td>
<td>1360</td>
</tr>
<tr>
<td>AxisUnit</td>
<td>1360</td>
</tr>
<tr>
<td>MajorTick</td>
<td>1361</td>
</tr>
<tr>
<td>MinorTick</td>
<td>1362</td>
</tr>
<tr>
<td>Transform</td>
<td>1362</td>
</tr>
<tr>
<td>TransformDate</td>
<td>1363</td>
</tr>
<tr>
<td>AxisR</td>
<td>1365</td>
</tr>
<tr>
<td>AxisRLabel</td>
<td>1367</td>
</tr>
<tr>
<td>AxisRLine</td>
<td>1368</td>
</tr>
<tr>
<td>AxisRMajorTick</td>
<td>1369</td>
</tr>
<tr>
<td>AxisTheta</td>
<td>1370</td>
</tr>
<tr>
<td>GridPolar</td>
<td>1372</td>
</tr>
<tr>
<td>Data</td>
<td>1372</td>
</tr>
<tr>
<td>ChartFunction</td>
<td>1383</td>
</tr>
<tr>
<td>ChartSpline</td>
<td>1384</td>
</tr>
<tr>
<td>Text</td>
<td>1385</td>
</tr>
<tr>
<td>ToolTip</td>
<td>1388</td>
</tr>
<tr>
<td>FillPaint</td>
<td>1390</td>
</tr>
</tbody>
</table>
AbstractChartNode Class

public class Imsl.Chart2D.AbstractChartNode

The base class of all of the nodes in 2D chart trees.
### Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Remarks</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUTOSCALE DATA</strong></td>
<td>An int that indicates autoscaling is to be done by scanning the data nodes.</td>
<td></td>
<td>Imsl.Chart2D.AbstractChartNode.AutoscaleInput (p. 1298)</td>
</tr>
<tr>
<td><strong>AUTOSCALE DENSITY</strong></td>
<td>An int that indicates autoscaling is to adjust the “Density” attribute.</td>
<td>This applies only to time axes.</td>
<td>Imsl.Chart2D.AbstractChartNode.AutoscaleOutput (p. 1299)</td>
</tr>
<tr>
<td><strong>AUTOSCALE NUMBER</strong></td>
<td>An int that indicates autoscaling is to adjust the “Number” attribute.</td>
<td></td>
<td>Imsl.Chart2D.AbstractChartNode.AutoscaleOutput (p. 1299)</td>
</tr>
<tr>
<td><strong>AUTOSCALE OFF</strong></td>
<td>An int that indicates autoscaling is turned off.</td>
<td></td>
<td>Imsl.Chart2D.AbstractChartNode.AutoscaleInput (p. 1298), Imsl.Chart2D.AbstractChartNode.AutoscaleOutput (p. 1299)</td>
</tr>
<tr>
<td><strong>AUTOSCALE WINDOW</strong></td>
<td>An int that indicates autoscaling is to be done by using the “Window” attribute.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **AXIS_X**             |                                                                                     |                                               |                                               |

---

**Chart2D**

AbstractChartNode • 1295
**Description**
An int that indicates the x-axis.
See Also: Type (p. 1356)

**AXIS_Y**
public int AXIS_Y

**Description**
An int that indicates the y-axis.
See Also: Type (p. 1356)

**DAY**
public int DAY

**Description**
An int which specifies a minimum tick mark interval for an autoscaled time axis where the time resolution is a day.

**HOUR**
public int HOUR

**Description**
An int which specifies a minimum tick mark interval for an autoscaled time axis where the time resolution is in hours.

**LABEL_TYPE_NONE**
public int LABEL_TYPE_NONE

**Description**
An int used to indicate that an element is not to be labeled.
See Also: Imsl.Chart2D.AbstractChartNode.LabelType (p. 1302)

**LABEL_TYPE_TITLE**
public int LABEL_TYPE_TITLE

**Description**
An int used to indicate that an element is to be labeled with the value of its title attribute.
See Also: Imsl.Chart2D.AbstractChartNode.LabelType (p. 1302)

**LABEL_TYPE_X**
public int LABEL_TYPE_X
**Description**
An int used to indicate that an element is to be labeled with the value of its x-coordinate.

See Also: Imsl.Chart2D.AbstractChartNode.LabelType (p. 1302)

**LABEL_TYPE_Y**
public int LABEL_TYPE_Y

**Description**
An int used to indicate that an element is to be labeled with the value of its y-coordinate.

See Also: Imsl.Chart2D.AbstractChartNode.LabelType (p. 1302)

**MILLISECOND**
public int MILLISECOND

**Description**
An int which specifies a minimum tick mark interval for an autoscaled time axis where the time resolution is in milliseconds.

**MINUTE**
public int MINUTE

**Description**
An int which specifies a minimum tick mark interval for an autoscaled time axis where the time resolution is in minutes.

**MONTH**
public int MONTH

**Description**
An int which specifies a minimum tick mark interval for an autoscaled time axis where the time resolution is a month.

**SECOND**
public int SECOND

**Description**
An int which specifies a minimum tick mark interval for an autoscaled time axis where the time resolution is in seconds.

**TRANSFORM_CUSTOM**
public int TRANSFORM_CUSTOM

**Description**
An int used to indicate that the axis using a custom transformation.

See Also: Imsl.Chart2D.AbstractChartNode.Transform (p. 1305)

**TRANSFORM_LINEAR**
public int TRANSFORM_LINEAR
Description
An int used to indicate that the axis uses linear scaling.
See Also: Imsl.Chart2D.AbstractChartNode.Transform (p. 1305)

TRANSFORM_LOG
public int TRANSFORM_LOG

Description
An int used to indicate that the axis uses logarithmic scaling.
See Also: Imsl.Chart2D.AbstractChartNode.Transform (p. 1305)

WEEK
public int WEEK

Description
An int which specifies a minimum tick mark interval for an autoscaled time axis where the time resolution is a week of the year.

YEAR
public int YEAR

Description
An int which specifies a minimum tick mark interval for an autoscaled time axis where the time resolution is a year.

Properties

AbstractParent
virtual public Imsl.Chart2D.AbstractChartNode AbstractParent {get; }

Description
Indicates the parent of this AbstractChartNode.

Property Value
An AbstractChartNode that is the parent of this node.

Remarks
If this is the root node in the chart tree the value is null.
Note that this is not an attribute setting.
Note that there is no SetParent method or property assignment.

AutoscaleInput
virtual public int AutoscaleInput {get; set; }

1298 • AbstractChartNode  IMSL C# Numerical Library
**Description**
Indicates what inputs are used for autoscaling.

**Property Value**
An int containing the “AutoscaleInput” attribute value.

**Remarks**
Legal values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSCALE_OFF</td>
<td>Do not do autoscaling.</td>
</tr>
<tr>
<td>AUTOSCALE_DATA</td>
<td>Use the data values. This is the default.</td>
</tr>
<tr>
<td>AUTOSCALE_WINDOW</td>
<td>Use the “Window” attribute value.</td>
</tr>
</tbody>
</table>

**AutoscaleMinimumTimeInterval**
virtual public int AutoscaleMinimumTimeInterval {get; set; }

**Description**
Specifies the minimum tick mark interval for autoscaled time axes.

**Property Value**
An int containing the “AutoscaleMinimumTimeInterval” attribute value.

**Remarks**
Legal values are:

- AbstractChartNode.MILLISECOND
- AbstractChartNode.SECOND
- AbstractChartNode.MINUTE
- AbstractChartNode.HOUR
- AbstractChartNode.DAY
- AbstractChartNode.WEEK
- AbstractChartNode.MONTH
- AbstractChartNode.YEAR

**AutoscaleOutput**
virtual public int AutoscaleOutput {get; set; }

**Description**
Specifies what attributes to change as a result of autoscaling.

**Property Value**
An int containing the “AutoscaleOutput” attribute value.
Remarks
Legal values are bitwise-or combinations of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSCALE_OFF</td>
<td>Do not do autoscaling.</td>
</tr>
<tr>
<td>AUTOSCALE_WINDOW</td>
<td>Change the “Window” attribute value.</td>
</tr>
<tr>
<td>AUTOSCALE_NUMBER</td>
<td>Change the “Number” attribute value.</td>
</tr>
<tr>
<td>AUTOSCALE_DENSITY</td>
<td>Change the “Density” attribute value.</td>
</tr>
</tbody>
</table>

By default, AutoscaleOutput = AUTOSCALE_NUMBER | AUTOSCALE_WINDOW | AUTOSCALE_DENSITY).

CultureInfo
virtual public System.Globalization.CultureInfo CultureInfo {get; set; }

Description
Adds support for Windows supported locales.

Property Value
A CultureInfo which contains the “CultureInfo” attribute.

Remarks
By default, CultureInfo = CurrentCulture.

CustomTransform
virtual public Imsl.Chart2D.Transform CustomTransform {get; set; }

Description
Allows for the specification of a custom transform.

Property Value
An Object implementing the Transform interface which specifies the “CustomTransform” attribute value.

Remarks
This is used only if the “Transform” attribute is set to TRANSFORM_CUSTOM.

Density
virtual public int Density {get; set; }

Description
Specifies the number of minor tick marks in the interval between major tick marks.

Property Value
An int containing the “Density” attribute value.
Remarks
By default, Density = 4.

FillColor
virtual public System.Drawing.Color FillColor {get; set; }

Description
Specifies a color that will be used to fill an area.

Property Value
A Color that contains the “FillColor” attribute value.

Remarks
By default, FillColor = Color.Black.

Font
virtual public System.Drawing.Font Font {get; set; }

Description
Defines a particular format for text, including font name, size, and style attributes.

Property Value
A Font that contains information about the text font to be used.

FontName
virtual public string FontName {get; set; }

Description
Specifies the font to be used by name.

Property Value
A String containing the “FontName” attribute value.

Remarks
By default, FontName = Sanserif.

FontSize
virtual public float FontSize {get; set; }

Description
Specifies the font size.

Property Value
A float containing the “FontSize” attribute value.

Remarks
By default, FontSize = 8.

FontStyle
virtual public System.Drawing.FontStyle FontStyle {get; set; }

Chart2D AbstractChartNode • 1301
Description
Specifies the font style to be used.

Property Value
A FontStyle containing the “FontStyle” attribute value.

Remarks
By default, FontStyle = FontStyle.Regular.

ImageAttr
virtual public System.Drawing.Image ImageAttr {get; set; }

Description
An image that is to be rendered when this ChartNode is displayed.

Property Value
An Image which contains the “Image” attribute value.

IsVisible
virtual public bool IsVisible {get; set; }

Description
Specifies if this node and its children will be drawn.

Property Value
A bool that contains the “IsVisible” attribute value.

Remarks
If false, this node and its children are not drawn. By default, IsVisible = true.

LabelType
virtual public int LabelType {get; set; }

Description
Specifies the type of label to display.

Property Value
An int which contains the “LabelType” attribute value.

Remarks
This indicates how a data point is to be labeled. The default is to not label data points, LABEL_TYPE_NONE.


LineColor
virtual public System.Drawing.Color LineColor {get; set; }

Description
Specifies the line color for this node.

Property Value
A Color which contains the “LineColor” attribute value.

Remarks
By default, LineColor = Color.Black.

LineWidth
virtual public double LineWidth {get; set; }

Description
Specifies the line width for this node.

Property Value
A double which contains the “LineWidth” attribute value.

Remarks
By default, LineWidth = 1.0.

MarkerColor
virtual public System.Drawing.Color MarkerColor {get; set; }

Description
Specifies what color will be used when rendering marker.

Property Value
A Color containing the “MarkerColor” attribute value.

Remarks
By default, MarkerColor = Color.Black.

MarkerSize
virtual public double MarkerSize {get; set; }

Description
Specifies the size of markers.

Property Value
A double which contains the “MarkerSize” attribute value.

Remarks
By default, MarkerSize = 1.0.

Name
virtual public string Name {get; set; }

Description
Specifies the name of this node.
**Property Value**
A String that contains the “Name” attribute value.

**Number**
virtual public int Number {get; set; }

**Description**
Specifies the number of tick marks along an axis.

**Property Value**
An int which contains the “Number” attribute value.

**Remarks**
By default, Number = 0.

**SkipWeekends**
virtual public bool SkipWeekends {get; set; }

**Description**
Specifies whether to skip weekends.

**Property Value**
A bool which contains the “SkipWeekends” attribute value.

**Remarks**
By default, SkipWeekends = false.

See Also: Imsl.Chart2D.AbstractChartNode.AutoscaleMinimumTimeInterval (p. 1299)

**SmoothingMode**
virtual public System.Drawing.Drawing2D.SmoothingMode SmoothingMode {get; set; }

**Description**
Specifies the SmoothingMode for this node.

**Property Value**
A SmoothingMode containing the “SmoothingMode” attribute value.

**Remarks**
By default, SmoothingMode = SmoothingMode.None (no antialiasing).

**TextColor**
virtual public System.Drawing.Color TextColor {get; set; }

**Description**
Specifies the text color.

**Property Value**
A Color containing the “TextColor” attribute value.
Remarks
The default value is Color.Black.

**TextFormat**

```csharp
virtual public string TextFormat {get; set; }
```

**Description**
Specifies the “TextFormat” attribute value.

**Property Value**
A string that contains the “TextFormat” attribute value.

**Remarks**
The default is “0.00” that allows exactly two digits after the decimal.

**TextFormatProvider**

```csharp
virtual public System.IFormatProvider TextFormatProvider {get; set; }
```

**Description**
Specifies the “TextFormatProvider” attribute value.

**Property Value**
An IFormatProvider that contains the “TextFormatProvider” attribute value.

**Remarks**
The default is null.

**TickLength**

```csharp
virtual public double TickLength {get; set; }
```

**Description**
This scales the length of the tick mark lines.

**Property Value**
A double which contains the “TickLength” attribute value.

**Remarks**
A value of 2.0 makes the tick marks twice as long as normal. A negative value causes the tick marks to be drawn pointing into the plot area. By default, TickLength = 1.0.

**Transform**

```csharp
virtual public int Transform {get; set; }
```

**Description**
Specifies whether the axis is linear, logarithmic or a custom transform.

**Property Value**
An int that contains the “Transform” attribute value.
Remarks
Legal values are Imsl.Chart2D.AbstractChartNode.TRANSFORM_LINEAR (p. 1297) (the default), Imsl.Chart2D.AbstractChartNode.TRANSFORM_LOG (p. 1298) and Imsl.Chart2D.AbstractChartNode.TRANSFORM_CUSTOM (p. 1297).

Constructor

AbstractChartNode
public AbstractChartNode(Imsl.Chart2D.AbstractChartNode parent)

Description
This interface contains members that will be common to chart objects in a variety of dimensions.

Parameter
parent – A chart node which is the parent node of this object.

Methods

GetAttribute
virtual public object GetAttribute(string name)

Description
Gets the value of an attribute.

Parameter
name – A String which specifies attribute that will have its value retrieved.

Returns
An Object which contains the specified attribute value.

GetBooleanAttribute
virtual public bool GetBooleanAttribute(string name, bool defaultValue)

Description
Convenience routine to get a Boolean-valued attribute.

Parameters
name – A String which contains the name of the attribute to be assessed.
defaultValue – A bool specifying the default value of the attribute.

Returns
A bool containing the attribute value.
Remarks
The value of an attribute is returned if it is defined and its value is of type bool. Otherwise the defaultValue is returned.

**GetColorAttribute**

virtual public System.Drawing.Color GetColorAttribute(string name)

Description
Convenience routine to get a Color-valued attribute.

**Parameter**

name – A String which contains the name of the attribute to be assessed.

**Returns**
A Color containing the attribute value.

Remarks
The value of an attribute is returned if it is defined and its value is of type Color. Otherwise the Color.Black is returned.

**GetDoubleAttribute**

virtual public double GetDoubleAttribute(string name, double defaultValue)

Description
Convenience routine to get a Double-valued attribute.

**Parameters**

name – A String which contains the name of the attribute to be assessed.

defaultValue – A double specifying the default value of the attribute.

**Returns**
A double containing the attribute value.

Remarks
The value of an attribute is returned if it is defined and its value is of type double. Otherwise the defaultValue is returned.

**GetIntegerAttribute**

virtual public int GetIntegerAttribute(string name, int defaultValue)

Description
Convenience routine to get an Integer-valued attribute.

**Parameters**

name – A String which contains the name of the attribute to be assessed.

defaultValue – An int specifying the default value of the attribute.

**Returns**
An int containing the attribute value.
Remarks
The value of an attribute is returned if it is defined and its value is of type int. Otherwise the defaultValue is returned.

GetStringAttribute
virtual public string GetStringAttribute(string name)

Description
Convenience routine to get a String-valued attribute.

Parameter
name – A String which contains the name of the attribute to be assessed.

Remarks
The attribute value is returned if it is defined and its value is of type String.

GetX
virtual public double[] GetX()

Description
Returns the “X” attribute value.

Returns
A double[] which contains the “X” attribute value.

GetY
virtual public double[] GetY()

Description
Returns the “Y” attribute value.

Returns
A double[] which contains the “Y” attribute value.

IsAncestorOf
virtual public bool IsAncestorOf(Imsl.Chart2D.AbstractChartNode node)

Description
Determines if this node is an ancestor of the argument node.

Parameter
node – An AbstractChartNode object that will have it’s relationship checked.
Returns
A bool, true if this node is an ancestor of the argument, node.

IsAttributeSet
virtual public bool IsAttributeSet(string name)

Description
Determines if an attribute is defined (may have been inherited).

Parameter
   name – A String which contains the name of the attribute.

Returns
A bool, true if the attribute is defined for this node. The definition may have been inherited from its parent node.

IsAttributeSetAtThisNode
virtual public bool IsAttributeSetAtThisNode(string name)

Description
Determines if an attribute is defined in this node (not inherited).

Parameter
   name – A String which contains the name of the attribute to be checked.

Returns
A bool, true if the attribute is defined in this node.

Remarks
The definition must have been set directly in this node, not just inherited from its parent node.

IsBitSet
static public bool IsBitSet(int flag, int mask)

Description
Returns true if the bit set in flag is set in mask.

Parameters
   flag – An int which contains the bit to be tested against the mask.
   mask – A int which is used as the mask.

Returns
A bool, true if the bit set in flag is set in mask.

Remove
public void Remove()
Description
Removes the node from its parents list of children.

setAttribute
virtual public void SetAttribute(string name, object value)

Description
Sets an attribute.

Parameters

  name – A String which contains the name of the attribute to be set.
  value – An Object which contains the attribute value.

SetX
virtual public void SetX(object x)

Description
Sets the “X” attribute value.

Parameter

  x – An Object that specifies the “X” attribute value.

SetY
virtual public void SetY(object y)

Description
Sets the “Y” attribute value.

Parameter

  y – An Object that specifies the “Y” attribute value.

ToString
override public string ToString()

Description
Returns the name of this chart node.

Returns
A String, the name of this chart node.

ChartNode Class

public class Imsl.Chart2D.ChartNode : AbstractChartNode
The base class of all of the nodes in the 2D chart tree.
Fields

**AXIS_X_TOP**

public int AXIS_X_TOP

**Description**
Flag to indicate x-axis placed on top of the chart.

**AXIS_Y_RIGHT**

public int AXIS_Y_RIGHT

**Description**
Flag to indicate y-axis placed to the right of the chart.

**BAR_TYPE_HORIZONTAL**

public int BAR_TYPE_HORIZONTAL

**Description**
Flag to indicate a horizontal bar chart.

See Also: Imsl.Chart2D.ChartNode.BarType (p. 1319)

**BAR_TYPE_VERTICAL**

public int BAR_TYPE_VERTICAL

**Description**
Flag to indicate a vertical bar chart.

See Also: Imsl.Chart2D.ChartNode.BarType (p. 1319)

**DASH_PATTERN_DASH**

public double[] DASH_PATTERN_DASH

**Description**
A double[] flag that specifies the rendering of a dashed line.

See Also: Imsl.Chart2D.ChartNode.SetLineDashPattern(System.Double[]) (p. 1330)

**DASH_PATTERN_DASH_DOT**

public double[] DASH_PATTERN_DASH_DOT

**Description**
A double[] flag that specifies the rendering of a dash-dot patterned line.

See Also: Imsl.Chart2D.ChartNode.SetLineDashPattern(System.Double[]) (p. 1330)

**DASH_PATTERN_DOT**

public double[] DASH_PATTERN_DOT
Description
A double[] flag that specifies the rendering of a dotted line.

See Also: Imsl.Chart2D.ChartNode.SetLineDashPattern(System.Double[]) (p. 1330)

DASH_PATTERN_SOLID
public double[] DASH_PATTERN_SOLID

Description
A double[] flag that specifies the rendering of a solid line.

See Also: Imsl.Chart2D.ChartNode.SetLineDashPattern(System.Double[]) (p. 1330)

DATA_TYPE_FILL
public int DATA_TYPE_FILL

Description
An int which when assigned to attribute “DataType” indicates that the area between the lines connecting data points and the horizontal reference line (y = attribute “Reference”) should be filled.

Remarks
This is an area chart.

DATA_TYPE_LINE
public int DATA_TYPE_LINE

Description
An int which when assigned to attribute “DataType” indicates that data points should be connected with line segments.

Remarks
This is the default setting.

DATA_TYPE_MARKER
public int DATA_TYPE_MARKER

Description
An int which when assigned to attribute “DataType” indicates that a marker should be drawn at each data point.

DATA_TYPE_PICTURE
public int DATA_TYPE_PICTURE

Description
An int which when assigned to attribute “DataType” indicates that an image (attribute “Image”) should be drawn at each data point.
Remarks
This can be used to draw fancy markers.

**DENDROGRAM_TYPE_HORIZONTAL**
public int DENDROGRAM_TYPE_HORIZONTAL

Description
Flag to indicate a horizontal dendrogram.

**DENDROGRAM_TYPE_VERTICAL**
public int DENDROGRAM_TYPE_VERTICAL

Description
Flag to indicate a vertical dendrogram.

**FILL_TYPE_GRADIENT**
public int FILL_TYPE_GRADIENT

Description
An int which indicates that a region will be drawn in a color gradient as specified by the attribute “Gradient”.

Remarks
This constant may be used with the Imsl.Chart2D.ChartNode.FillType (p. 1322) property.

See Also:

**FILL_TYPE_NONE**
public int FILL_TYPE_NONE

Description
An int which indicates that a region is not to be drawn.

Remarks
When Imsl.Chart2D.ChartNode.FillType (p. 1322) and Imsl.Chart2D.ChartNode.FillOutlineType (p. 1321) are set to this value the region will not be rendered

**FILL_TYPE_PAINT**
public int FILL_TYPE_PAINT

Description
An int which indicates that a region will be drawn using the texture specified by the “FillPaint” attribute.


**FILL_TYPE_SOLID**
public int FILL_TYPE_SOLID
Description
An int which indicates that a region will be drawn using the solid color specified by
Imsl.Chart2D.ChartNode.FillType (p. 1322) and Imsl.Chart2D.ChartNode.FillOutlineType (p. 1321).

**LABEL_TYPE_PERCENT**

public int LABEL_TYPE_PERCENT

Description
An int which indicates that a pie slice is to be labeled with a percentage value.

Remarks
This flag only applies to pie charts.
See Also: LabelType (p. 1302)

**MARKER_TYPE_ASTERISK**

public int MARKER_TYPE_ASTERISK

Description
An int that indicates an asterisk is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

**MARKER_TYPE_CIRCLE_CIRCLE**

public int MARKER_TYPE_CIRCLE_CIRCLE

Description
An int that indicates a circle in a circle is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

**MARKER_TYPE_CIRCLE_PLUS**

public int MARKER_TYPE_CIRCLE_PLUS

Description
An int that indicates an plus in a circle is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

**MARKER_TYPE_CIRCLE_X**

public int MARKER_TYPE_CIRCLE_X

Description
An int that indicates an x in a circle is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

**MARKER_TYPE_DIAMOND_PLUS**

public int MARKER_TYPE_DIAMOND_PLUS
Description
An int that indicates a plus in a diamond is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)
MARKER_TYPE_FILLED_CIRCLE
public int MARKER_TYPE_FILLED_CIRCLE

Description
An int that indicates a filled circle is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)
MARKER_TYPE_FILLED_DIAMOND
public int MARKER_TYPE_FILLED_DIAMOND

Description
An int that indicates a filled diamond is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)
MARKER_TYPE_FILLED_SQUARE
public int MARKER_TYPE_FILLED_SQUARE

Description
An int that indicates a filled square is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)
MARKER_TYPE_FILLED_TRIANGLE
public int MARKER_TYPE_FILLED_TRIANGLE

Description
An int that indicates a filled triangle is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)
MARKER_TYPE_HOLLOW_CIRCLE
public int MARKER_TYPE_HOLLOW_CIRCLE

Description
An int that indicates a hollow circle is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)
MARKER_TYPE_HOLLOW_DIAMOND
public int MARKER_TYPE_HOLLOW_DIAMOND
Description
An int that indicates a hollow diamond is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

MARKER_TYPE_HOLLOW_SQUARE
public int MARKER_TYPE_HOLLOW_SQUARE

Description
An int that indicates a hollow square is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

MARKER_TYPE_HOLLOW_TRIANGLE
public int MARKER_TYPE_HOLLOW_TRIANGLE

Description
An int that indicates a hollow triangle is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

MARKER_TYPE_OCTAGON_PLUS
public int MARKER_TYPE_OCTAGON_PLUS

Description
An int that indicates a plus in an octagon is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

MARKER_TYPE_OCTAGON_X
public int MARKER_TYPE_OCTAGON_X

Description
An int that indicates a x in an octagon is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

MARKER_TYPE_PLUS
public int MARKER_TYPE_PLUS

Description
An int that indicates a plus-shaped data marker is to be drawn.

Remarks
This is the default value of Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

MARKER_TYPE_SQUARE_PLUS
public int MARKER_TYPE_SQUARE_PLUS
Description
An int that indicates a x in a square is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

**MARKER_TYPE_SQUARE_X**
public int MARKER_TYPE_SQUARE_X

Description
An int that indicates a x in a diamond is to be drawn as the data marker.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

**MARKER_TYPE_X**
public int MARKER_TYPE_X

Description
An int that indicates a x-shaped data marker is to be drawn.
See Also: Imsl.Chart2D.ChartNode.MarkerType (p. 1323)

**TEXT_X_CENTER**
public int TEXT_X_CENTER

Description
An int which indicates that text should be centered.
See Also: Alignment (p. 1386)

**TEXT_X_LEFT**
public int TEXT_X_LEFT

Description
An int which indicates that text should be left justified.
See Also: Alignment (p. 1386)

**TEXT_X_RIGHT**
public int TEXT_X_RIGHT

Description
An int which indicates that text should be right justified.
See Also: Alignment (p. 1386)

**TEXT_Y_BOTTOM**
public int TEXT_Y_BOTTOM
Description
An int which indicates that text should be drawn on the baseline.
See Also: Alignment (p. 1386)

**TEXT_Y_CENTER**

public int TEXT_Y_CENTER

**Description**
An int which indicates that text should be vertically centered.
See Also: Alignment (p. 1386)

**TEXT_Y_TOP**

public int TEXT_Y_TOP

**Description**
An int which indicates that text should be drawn with the top of the letters touching the top of the drawing region.
See Also: Alignment (p. 1386)

**WebCtrl**

protected internal bool WebCtrl

**Description**
A bool indicating if this ChartNode is a WebControl.

**Properties**

**ALT**

virtual public string ALT {get; set; }

**Description**
Used to construct an “alt” attribute value in client side image maps.

**Property Value**
A String which contains the “ALT” attribute value.

**Remarks**
The “alt” attribute is used when client-side image maps are generated. A client-side image map has an entry for each node in which the chart attribute HREF is defined. Some browsers use the alt tag value as tooltip text.

**Axis**

virtual public Imsl.Chart2D.Axis Axis {get; }

1318 • ChartNode

IMSL C# Numerical Library
Description
Typically provides a mapping for children from the user coordinate space to the device (screen) space.

Property Value
An Axis that contains the “Axis” attribute value.

Background
virtual public Imsl.Chart2D.Background Background {get; }

Description
The base graphic layer displayed behind other ChartNode objects in the tree.

Property Value
A Background which contains the “Background” attribute value.

BarGap
virtual public double BarGap {get; set; }

Description
Specifies the gap between bars in a group.

Property Value
A double which contains the “BarGap” attribute value.

Remarks
A gap of 1.0 means that space between bars is the same as the width of an individual bar in the group. By default, BarGap = 0.0.

BarType
virtual public int BarType {get; set; }

Description
Specifies the orientation of the BarChart.

Property Value
An int which contains the “BarType” attribute value.

Remarks
Legal values are Imsl.Chart2D.ChartNode.BAR_TYPE_VERTICAL (p. 1311) or Imsl.Chart2D.ChartNode.BAR_TYPE_HORIZONTAL (p. 1311).

BarWidth
virtual public double BarWidth {get; set; }

Description
The width of all of the groups of bars at each index.

Property Value
A double which contains the “BarWidth” attribute value.
Remarks
By default, BarWidth = 0.5.

Chart
virtual public Imsl.Chart2D.Chart Chart {get; }

Description
This is the root node of the chart tree.

Property Value
A Chart that contains the “Chart” attribute value.

ChartTitle
virtual public Imsl.Chart2D.ChartTitle ChartTitle {get; set; }

Description
Specifies a title for the chart.

Property Value
A ChartTitle which contains the “ChartTitle” attribute value.

Remarks
This is effective only in the ChartNode, where it replaces the existing ChartTitle node.

ClipData
virtual public bool ClipData {get; set; }

Description
Specifies whether the data elements are to be clipped to the current window.

Property Value
A bool which contains the “ClipData” attribute value.

Remarks
By default, ClipData = true

DataType
virtual public int DataType {get; set; }

Description
Specifies how the data is to be rendered.

Property Value
An int that contains the “DataType” attribute value.
Remarks
This should be some xor-ed combination of Imsl.Chart2D.ChartNode.DATA_TYPE_LINE (p. 1312), Imsl.Chart2D.ChartNode.DATA_TYPE_MARKER (p. 1312). By default, DataType = DATA_TYPE_LINE (p. 1312).

**DoubleBuffering**

virtual public bool DoubleBuffering {get; set; }

**Description**
Specifies whether double is active.

**Property Value**
A bool which contains the “DoubleBuffering” attribute value.

**Remarks**
Double buffering reduces flicker when the screen is updated. This attribute only has an effect if it is set at the root node of the chart tree.

**Explode**

virtual public double Explode {get; set; }

**Description**
Specifies how far from the center pie slices are drawn.

**Property Value**
A double containing the “Explode” attribute value.

**Remarks**
The scale is proportional to the pie chart’s radius. By default, Explode = 0.0.

**FillOutlineColor**

virtual public System.Drawing.Color FillOutlineColor {get; set; }

**Description**
Specifies a color that will be used to outline this node.

**Property Value**
A Color which contains the “FillOutlineColor” attribute value.

**Remarks**
The default value is Color.Black.

**FillOutlineType**

virtual public int FillOutlineType {get; set; }

**Description**
Specifies a fill pattern type for the outline of this node.

**Property Value**
An int that contains the “FillOutlineType” attribute value.
Remarks
By default, FillOutlineType = FILL_TYPE_SOLID (p. 1313).

FillType
virtual public int FillType {get; set; }

Description
Specifies a fill pattern type for this node.

Property Value
An int which contains the “FillType” attribute value.

Remarks
By default, FillType = FILL_TYPE_SOLID (p. 1313).
See Also: Imsl.Chart2D.ChartNode.FILL_TYPE_NONE (p. 1313),
Imsl.Chart2D.ChartNode.FILL_TYPE_GRADIENT (p. 1313),
Imsl.Chart2D.ChartNode.FILL_TYPE_PAINT (p. 1313)

HREF
virtual public string HREF {get; set; }

Description
Used to specify an “activated” object in an image map.

Property Value
A String which contains the “HREF” attribute value.

Remarks
The “HREF” attribute is used when client-side image maps are generated. A client-side image map has
an entry for each node in which the chart attribute HREF is defined. The values of HREF attributes are
URLs. Such regions treated by the browser as hyperlinks.

ImageAttr
virtual public System.Drawing.Image ImageAttr {get; set; }

Description
An image that is to rendered when this ChartNode is displayed.

Property Value
An Image which contains the “Image” attribute value.

IsWebControl
public bool IsWebControl {get; }

Description
Indicates whether this is a web control.
**Property Value**
A bool which is true if the node is a WebControl.

**Legend**
virtual public Imsl.Chart2D.Legend Legend {get; }

**Description**
Legend information associated with this ChartNode.

**Property Value**
A Legend containing legend information associated with this ChartNode.

**MarkerThickness**
virtual public double MarkerThickness {get; set; }

**Description**
Specifies the line thickness to be used when rendering the markers.

**Property Value**
A double that contains the “MarkerThickness” attribute value.

**Remarks**
If “MarkerThickness” is 2.0 then markers are drawn twice as thick as normal. By default, MarkerThickness = 1.0.

**MarkerType**
virtual public int MarkerType {get; set; }

**Description**
Specifies the type of data marker to be drawn.

**Property Value**
The int which contains the “MarkerType” attribute value.

**Remarks**
By default, MarkerType = Imsl.Chart2D.ChartNode.MARKER_TYPE_PLUS (p. 1316).

See Also:  Imsl.Chart2D.ChartNode.MARKER_TYPE_ASTERISK (p. 1314),
Imsl.Chart2D.ChartNode.MARKER_TYPE_X (p. 1317),
Imsl.Chart2D.ChartNode.MARKER_TYPE_HOLLOW_SQUARE (p. 1316),
Imsl.Chart2D.ChartNode.MARKER_TYPE_HOLLOW_TRIANGLE (p. 1316),
Imsl.Chart2D.ChartNode.MARKER_TYPE_HOLLOW_DIAMOND (p. 1315),
Imsl.Chart2D.ChartNode.MARKER_TYPE_DIAMOND_PLUS (p. 1316),
Imsl.Chart2D.ChartNode.MARKER_TYPE_DIAMOND (p. 1316),
Imsl.Chart2D.ChartNode.MARKER_TYPE_OCTAGON_X (p. 1316),
Imsl.Chart2D.ChartNode.MARKER_TYPE_OCTAGON_PLUS (p. 1316),
Imsl.Chart2D.ChartNode.MARKER_TYPE_OCTAGON (p. 1316),
Imsl.Chart2D.ChartNode.MARKER_TYPE_HOLLOW_CIRCLE (p. 1315),
Imsl.Chart2D.ChartNode.MARKER_TYPE_CIRCLE_X (p. 1314),
Parent

virtual public Imsl.Chart2D.ChartNode Parent {get; }

Description
Indicates the parent of this ChartNode.

Property Value
A ChartNode object which contains this node’s parent.

Remarks
This is null in the case of the root node of the chart tree, since that node has no parent.
Note that this is not an attribute setting.
Note that there is no function to set the Parent.

Reference

virtual public double Reference {get; set; }

Description
Indicates the baseline in drawing area charts.

Property Value
A double which contains the “Reference” attribute value.

Remarks
In the case of a pie chart, this specifies the angle (in degrees) of the first slice. By default, Reference = 0.0.

ScreenAxis

virtual public Imsl.Chart2D.AxisXY ScreenAxis {get; }

Description
Provides a default mapping from the user coordinates [0,1] by [0,1] to the screen.

Property Value
An AxisXY that contains the “ScreenAxis” attribute value.

Remarks
This is set by the root ChartNode, so there is no set ScreenAxis accessor.
See Also: Imsl.Chart2D.Chart (p. 1332)

ScreenState

virtual public System.Drawing.Size ScreenSize {get; set; }

1324 • ChartNode  IMSL C# Numerical Library
**Description**
Indicates the size of this ChartNode.

**Property Value**
A Size that contains the “ScreenSize” attribute value.

**Remarks**
If this attribute has not been defined the size of the “Control” attribute is returned. If neither attribute is defined null is returned.

---

**Size**
virtual public System.Drawing.Size Size {get; set; }

**Description**
Specifies the drawing size.

**Property Value**
A Size which contains the “Size” attribute value.

---

**TextAngle**
virtual public int TextAngle {get; set; }

**Description**
An angle, in degrees, at which text is to be drawn.

**Property Value**
An int that contains the angle at which text is to be drawn.

**Remarks**
Only multiples of 90 are allowed at this time. By default, TextAngle = 0.

---

**ToolTip**
virtual public string ToolTip {get; set; }

**Description**
Text that can be displayed in the case where tool tips are used.

**Property Value**
A String which contains the “ToolTip” attribute value.

---

**Constructor**

**ChartNode**
public ChartNode(Imsl.Chart2D.ChartNode parent)

**Description**
Constructs a ChartNode object.
Parameter

- parent – The ChartNode which is the parent of this object.

Methods

**FirePickListeners**

```csharp
virtual public void FirePickListeners(System.Windows.Forms.MouseEventArgs e)
```

**Description**

Invokes the pick delegates defined at this node and at all of its ancestors, if the event “hits” the node.

**Parameter**

- `e` – A MouseEventArgs which determines which nodes have been selected.

**GetChildren**

```csharp
virtual public Imsl.Chart2D.ChartNode[] GetChildren()
```

**Description**

Gets the list of child nodes.

**Returns**

- A `ChartNode[]` which contains the children of this node.

**Remarks**

If there are no children, a 0-length array is returned.

**GetComponent**

```csharp
virtual public System.Windows.Forms.Control GetComponent()
```

**Description**

Gets the “Component” attribute value.

**Returns**

- A Control that contains the “Component” attribute value.

**GetConcatenatedViewport**

```csharp
virtual public double[] GetConcatenatedViewport()
```

**Description**

Returns the value of the “Viewport” attribute concatenated with the “Viewport” attributes set in its ancestor nodes.

**Returns**

Remarks
By default, GetConcatenatedViewport = \{0.0, 1.0, 0.0, 1.0\}.

GetFillPaint
virtual public System.Drawing.Brush GetFillPaint()

Description
Returns the “FillPaint” attribute value.

Returns
The value of the “FillPaint” attribute, if defined. Otherwise, null is returned.

GetGradient
virtual public System.Drawing.Color[] GetGradient()

Description
Returns the value of the “Gradient” attribute.

Returns

Remarks
The array is of length four, containing \{colorLL, colorLR, colorUR, colorUL\}. By default, GetGradient = null.

GetLineDashPattern
virtual public double[] GetLineDashPattern()

Description
Returns the “LineDashPattern” attribute value.

Returns
A double[] that contains the line “LineDashPattern” attribute value.

Remarks
Returns null if the attribute has not been defined.

GetMarkerDashPattern
virtual public double[] GetMarkerDashPattern()

Description
Returns the “MarkerDashPattern” attribute value.

Returns
A double[] that contains the “MarkerDashPattern” attribute value.

Remarks
Returns null if the attribute has not been defined.

GetScreenViewport
virtual public int[] GetScreenViewport()
**GetViewport**

virtual public double[] GetViewport()

**Description**

Returns the value of the “Viewport” attribute.

**Returns**


**Remarks**

By default, GetViewport = {0.0, 1.0, 0.0, 1.0}.

**GetWebComponent**


**Description**

Gets the “Component” attribute value.

**Returns**

A WebControl that contains the “Component” attribute value.

**IsBitSet**

static public bool IsBitSet(int flag, int mask)

**Description**

Determins if the bit set in flag is set in mask.

**Parameters**

- flag – An int which contains the bit to be tested against mask.
- mask – An int which is to be used as teh mask.
Returns
A bool which is true if the bit set in flag is set in mask.

OnPick
void OnPick(Imsl.Chart2D.PickEventArgs eventParam)

Description
Invokes delegates registered with the Pick event.

Parameter
  eventParam – A PickEventArgs that specifies the event data.

Paint
abstract public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
  draw – A Draw which is to be painted.

PrePaint
virtual public void PrePaint()

Description
The PrePaint method is called in all nodes in a chart just before the chart is painted. The default implementation does nothing. Override this method to do computations just before painting in a chart node.

SetFillPaint
virtual public void SetFillPaint(System.Drawing.Brush brush)

Description
Sets the value of the “FillPaint” attribute.

Parameter
  brush – A Brush which specifies the “FillPaint” attribute value.

SetFillPaint
virtual public void SetFillPaint(System.Drawing.Image imageIcon)

Description
Sets the “FillPaint” attribute value.

Parameter
  imageIcon – A Image that specifies the “FillPaint” attribute value.

SetFillPaint
virtual public void SetFillPaint(System.Uri uriImage)
Description
Sets the “FillPaint” attribute value.

Parameter
uriImage – A Uri which specifies the location of an image used to set the “FillPaint” attribute.

SetGradient

Description
Sets the value of the “Gradient” attribute.

Parameters
- colorLL – A Color value which specifies the color of the lower left corner.
- colorLR – A Color value which specifies the color of the lower right corner.
- colorUR – A Color value which specifies the color of the upper right corner.
- colorUL – A Color value which specifies the color of the upper left corner.

Remarks
This attribute defines a color gradient used to fill regions. Only two of the four colors given are actually used.

<table>
<thead>
<tr>
<th>Parameter Values</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>colorLL==colorLR and colorUL==colorUR</td>
<td>A vertical gradient is drawn.</td>
</tr>
<tr>
<td>colorLL==colorUL and colorLR==colorUR</td>
<td>A horizontal gradient is drawn.</td>
</tr>
<tr>
<td>colorLR==null and colorUL==null</td>
<td>A diagonal gradient is drawn.</td>
</tr>
<tr>
<td>colorLL==null and colorUR==null</td>
<td>A diagonal gradient is drawn.</td>
</tr>
</tbody>
</table>

SetGradient
virtual public void SetGradient(System.Drawing.Color[] colorGradient)

Description
Sets the value of the “Gradient” attribute.

Parameter
colorGradient – A Color[4] containing the colors at the lower left, lower right, upper right and upper left corners of the bounding box of the regions being filled.

SetLineDashPattern
virtual public void SetLineDashPattern(double[] lineDashPattern)

Description
Sets the “LineDashPattern” attribute value.
Parameter

    lineDashPattern – A double[] which specifies the line dash pattern to be rendered.

SetMarkerDashPattern
virtual public void SetMarkerDashPattern(double[] markerDashPattern)

Description
Sets the “MarkerDashPattern” attribute value.

Parameter

    markerDashPattern – A double[] that specifies the “MarkerDashPattern” attribute value.

SetTitle
virtual public void SetTitle(string title)

Description
Sets the value of the “Title” attribute.

Parameter

    title – A String which specifies the “Title” attribute value.

SetTitle
virtual public void SetTitle(Imsl.Chart2D.Text title)

Description
Sets the value of the “Title” attribute.

Parameter

    title – A Text which specifies the “Title” attribute value.

SetViewport
virtual public void SetViewport(double[] viewport)

Description
Used to specify the viewport location.

Parameter


Remarks
The viewport is the subregion of the drawing surface where the plot is to be drawn. “Viewport”
coordinates are [0,1] by [0,1] with (0,0) in the upper left corner. The “Viewport” attribute affects only
Axis nodes, since they contain the mappings to device space. The elements of viewport correspond to
xmin, xmax, ymin and ymax.

SetViewport
virtual public void SetViewport(double xmin, double xmax, double ymin, double ymax)
**Description**

Used to specify the viewport location.

**Parameters**

- *xmin* – A *double* specifying the left side of the viewport.
- *xmax* – A *double* specifying the right side of the viewport.
- *ymin* – A *double* specifying the bottom of the viewport.
- *ymax* – A *double* specifying the top of the viewport.

**Remarks**

The viewport is the subregion of the drawing surface where the plot is to be drawn. “Viewport” coordinates are [0,1] by [0,1] with (0,0) in the lower left corner. The “Viewport” attribute affects only Axis nodes, since they contain the mappings to device space.

---

**Chart Class**

```csharp
public class Imsl.Chart2D.Chart : ChartNode : ICloneable

The root node of the chart tree.

This chart node creates the following child nodes: Imsl.Chart2D.Background (p. 1338), Imsl.Chart2D.ChartTitle (p. 1339) and Imsl.Chart2D.Legend (p. 1341).
```

**Constructors**

**Chart**

```csharp
public Chart()

Description

This is the root of our tree, it has no parent.

Remarks

This creates the Chart with a null component.
```

**Chart**

```csharp
public Chart(System.Windows.Forms.Control component)

Description

This is the root of our tree, it has no parent.

Parameter

- **component** – A Component that contains the chart.
```
Remarks
This creates the Chart with the named Component.

Chart
public Chart(System.Web.UI.WebControls.WebControl component)

Description
This is the root of our tree, it has no parent.

Parameter
    component – This creates the Chart with the named WebControl.

Chart
public Chart(System.Drawing.Image image)

Description
This is the root of our tree, it has no parent.

Parameter
    image – An Image into which the Chart is to be drawn.

Remarks
This creates the Chart drawn into the Image.

Methods

AddLegendItem
virtual public void AddLegendItem(int type, Imsl.Chart2D.ChartNode node)

Description
Adds a Legend to a ChartNode.

Parameters
    type – An int which specifies the LegendItem type.
    node – A ChartNode to which a Legend is to be added.

Remarks
This method is intended to be called from within the Paint method of classes which explicitly paint their own child chart nodes. The child chart nodes to be included in the legend must be added to the legend during each call to Paint of the parent chart node.

Typical users of the chart library do not need to call this routine. This method is for use by those writing new charting classes.

The possible legend types are:

Chart2D
Clone
virtual public object Clone()

Description
Returns a clone of the graphics tree.

Returns
An Object which is a clone of this graphics tree.

Copy
virtual public void Copy()

Description
Copy the chart to the clipboard.

Finalize
override void Finalize()

Description
Finalize disposes the image buffer.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

Paint
virtual public void Paint(System.Drawing.Graphics g)

Description
Paints this node and all of its children.

Parameter
g – A Graphics which is to be painted.
Remarks
This is normally called only by the Paint method in this node’s parent.

**PaintChart**
virtual public void PaintChart(System.Drawing.Graphics graphics)

Description
Draw the chart using the given Graphics object.

Parameter

**PaintImage**
virtual public System.Drawing.Image PaintImage()

Description
Returns an Image of the chart.

Returns
An Image containing a picture of the chart.

**Pick**
virtual public void Pick(System.Windows.Forms MouseEventArgs mouseEvent)

Description
Invoke the pick delegates for the nodes hit by the event.

Parameter
    mouseEvent – A MouseEventArgs whose position determines which nodes have been selected.

**PrintGraphics**

Description
This method prints the chart on a single page.

Parameters
    sender – A Object that specifies the sender of an event.
    e – A PrintPageEventArgs containing data for the PrintPage event.

Remarks
The output is scaled to fill the page as much as possible while preserving the aspect ratio.

**Repaint**
virtual public void Repaint()
**Description**
Prepares the chart to be repainted by deleting any double buffering image.

**SetComponent**
```
virtual public void SetComponent(System.Windows.Forms.Control component)
```

**Description**
Sets the “Component” attribute value.

**Parameter**
- `component` – A `Control` that contains a component of the Chart.

**Update**
```
virtual public void Update(System.Drawing.Graphics g)
```

**Description**
Update the chart when its component is damaged. The component can be damaged when the window containing it is iconified or is overlapped by another window.

**Parameter**
- `g` – is a `Graphics` object used for drawing.

**WritePNG**
```
virtual public void WritePNG(System.IO.Stream os, int width, int height)
```

**Description**
Writes the chart as an PNG file.

**Parameters**
- `os` – A `Stream` containing the output stream to which the PNG image is to be written.
- `width` – An `int` which specifies the width of the output image in pixels.
- `height` – An `int` which specifies the height of the output image in pixels.

**Example: Area Chart**
An area chart is constructed in this example. Three data sets are used and a legend is added to the chart.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class AreaEx1 : FrameChart
{
    public AreaEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);
        int npoints = 20;
    }
}
```
double dx = .5 * System.Math.PI / (npoints - 1);
double[] x = new double[npoints];
double[] y1 = new double[npoints];
double[] y2 = new double[npoints];
double[] y3 = new double[npoints];

// Generate some data
for (int i = 0; i < npoints; i++)
{
    x[i] = i * dx;
    y1[i] = System.Math.Sin(x[i]);
    y2[i] = System.Math.Cos(x[i]);
    y3[i] = System.Math.Atan(x[i]);
}
Data d1 = new Data(axis, x, y1);
Data d2 = new Data(axis, x, y2);
Data d3 = new Data(axis, x, y3);

// Set Data Type to Fill Area
axis.DataType = Imsl.Chart2D.Data.DATA_TYPE_FILL;

// Set Line Colors

// Set Fill Colors

// Set Data Labels
d1.SetTitle("Sine");
d2.SetTitle("Cosine");
d3.SetTitle("ArcTangent");

// Add a Legend
Legend legend = chart.Legend;
legend.SetTitle(new Text("Legend"));
legend.IsVisible = true;

// Set the Chart Title
chart.ChartTitle.SetTitle("Area Plots");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new AreaEx1());
}
Background Class

public class Imsl.Chart2D.Background : AxisXY

The background of a chart.

Grid is created by Imsl.Chart2D.Chart (p. 1332) as its child. It can be retrieved using the method Imsl.Chart2D.ChartNode.Background (p. 1319).

Fill attributes (specified with FillType (p. 1322)) in this node control the drawing of the background.
Method

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

ChartTitle Class

public class Imsl.Chart2D.ChartTitle : AxisXY

The main title of a chart.

ChartTitle is created by Chart (p. 1332) as its child. It can be retrieved using the method ChartTitle (p. 1339).

The axis title is the “Title” attribute value at this node. Text attributes (specified with CultureInfo, NumberFormatInfo.CurrentInfo and DateTimeFormatInfo.CurrentInfo members) in this node control the drawing of the title.

Method

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.
Grid Class

public class Imsl.Chart2D.Grid : ChartNode

Draws the grid lines perpendicular to an axis.

Grid is created by Imsl.Chart2D.Axis1D (p. 1353) as its child. It can be retrieved using the Imsl.Chart2D.Axis1D.Grid (p. 1355) property.

Line attributes (specified with LineColor (p. 1302), LineWidth (p. 1303) and SetMarkerDashPattern (p. 1331)) in this node control the drawing of the grid lines.

Property

Type

virtual public int Type {get; }

Description
Specifies the type of Axis1D.

Property Value
An int which contains the type of axis this Grid is associated with.

Remarks
The Axis types are:

- Imsl.Chart2D.AbstractChartNode.AXIS_X (p. 1295)
- Imsl.Chart2D.AbstractChartNode.AXIS_Y (p. 1296)
- Imsl.Chart2D.ChartNode.AXIS_X_TOP (p. 1311)
- Imsl.Chart2D.ChartNode.AXIS_Y_RIGHT (p. 1311)

Method

Paint

override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter

draw – A Draw which is to be painted.
Remarks
This is normally called only by the Paint method in this node’s parent.

Legend Class

public class Imsl.Chart2D.Legend : AxisXY
A Imsl.Chart2D.Chart (p. 1332) legend. Legend is created by Chart as its child. It can be retrieved using the Imsl.Chart2D.ChartNode.Legend (p. 1323) property.
By default the legend is not drawn. To have it drawn, set chart.IsVisible = true;
Imsl.Chart2D.Data (p. 1372) objects that have their “Title” attribute defined are automatically entered into the legend.
The “Viewport” attribute for this node is set to [0.83,0.98] by [0.1,0.6].
The drawing of the background of the legend box is controlled by the Fill attributes (specified with FillType (p. 1322)) in this node. Text attributes (specified with CultureInfo , NumberFormatInfo.CurrentInfo and DateTimeFormatInfo.CurrentInfo members) in this node control the drawing of the text strings in the box.

Method

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.
Annotation Class

public class Imsl.Chart2D.Annotation : Data

Draws an annotation.

Locations are defined in chart (user) coordinates. To convert between device and user coordinates see Axis.MapDeviceToUser (p. 1347) and Axis.MapUserToDevice (p. 1347).

Properties

String
public string String {get; set; }

Description
Sets the string for the Text object to render.

Property Value
The String for the Text object to render.

Text
public Imsl.Chart2D.Text Text {get; set; }

Description
The Text for this Annotation instance.

Property Value
A Text object

Constructors

Annotation
public Annotation(Imsl.Chart2D.ChartNode parent, Imsl.Chart2D.Text text, double x, double y)

Description
Creates a Text object at the specific x,y location in chart coordinates.

Parameters
  parent – The ChartNode parent of this data node, usually an Axis object.
  text – The Text object to draw.
Annotation
public Annotation(Imsl.Chart2D.ChartNode parent, string s, double x, double y)

Description
Creates a string object at the specific x,y location in chart coordinates.

Parameters
- parent – The ChartNode parent of this data node, usually an Axis object.
- s – The string object to draw.
- x – The x location in user coordinates.
- y – The y location in user coordinates.

Annotation
public Annotation(Imsl.Chart2D.ChartNode parent, System.Drawing.Image img, double x, double y)

Description
Renders an Image object centered at an x,y location in chart coordinates.

Parameters
- parent – The ChartNode parent of this data node, usually an Axis object.
- img – The Image object to draw.
- x – The x location in user coordinates.
- y – The y location in user coordinates.

Methods

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw object which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetLocation
public void SetLocation(double x, double y)
Description
Update the location of this Annotation instance.

Parameters
- x – A double specifying the new x location in chart coordinates.
- y – A double specifying the new y location in chart coordinates.

Example: Line Chart
A simple line chart is constructed in this example. Three data sets are used and a legend is added to the chart. An annotation is included indicating a feature on the chart.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class LineEx1 : FrameChart
{
    public LineEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
        double dx = .5 * System.Math.PI / (npoints - 1);
        double[] x = new double[npoints];
        double[] y1 = new double[npoints];
        double[] y2 = new double[npoints];
        double[] y3 = new double[npoints];

        // Generate some data
        for (int i = 0; i < npoints; i++)
        {
            x[i] = i * dx;
            y1[i] = System.Math.Sin(x[i]);
            y2[i] = System.Math.Cos(x[i]);
            y3[i] = System.Math.Atan(x[i]);
        }
        Data d1 = new Data(axis, x, y1);
        Data d2 = new Data(axis, x, y2);
        Data d3 = new Data(axis, x, y3);

        // Set Data Type to Line
        axis.DataType = Imsl.Chart2D.AxisXY.DATA_TYPE_LINE;

        // Set Line Colors

        // Set Data Labels
```
d1.SetTitle("Sine");
d2.SetTitle("Cosine");
d3.SetTitle("ArcTangent");

    // Add a Legend
    Legend legend = chart.Legend;
    legend.SetTitle(new Text("Legend"));
    legend.IsVisible = true;

    // Add an Annotation
    Text label = new Text("Maximum Values");
    label.Alignment = Axis.TEXT_X_RIGHT;
    Annotation ann = new Annotation(axis, label, 1.6, 1.02);

    // Set the Chart Title
    chart.ChartTitle.SetTitle("Line Plots");
    
public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new LineEx1());
}
}
Axis Class

public class Imsl.Chart2D.Axis : ChartNode

The Axis node provides the mapping for all of its children from the user coordinate space to the device (screen) space.
Constructor

Axis
Axis(Imsl.Chart2D.Chart chart)

Description
Constructs an Axis node.

Parameter
chart – A Chart object which is the parent of this node.

Remarks
The parent must be a Chart node. This node’s “Axis” attribute has itself as a value, so that decendent nodes can easily obtain their controlling axis node.

Methods

MapDeviceToUser
abstract public void MapDeviceToUser(int devX, int devY, double[] userXY)

Description
Maps the device coordinates to user coordinates.

Parameters
devX – An int which specifies the device x-coordinate.
devY – An int which specifies the device y-coordinate.
userXY – An int[2] array on input, on output, the user coordinates.

MapToDevice
abstract public void MapToDevice(double userX, double userY, int[] devXY)

Description
Maps the user coordinates (userX, userY) to the device coordinates devXY.

Parameters
userX – A double which specifies the user x-coordinate.
userY – A double which specifies the user y-coordinate.
devXY – An int[2] array on input, on output, the device coordinates.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.
Parameter

draw – A Draw object which specifies the chart tree to be rendered on the screen.

Remarks

This is normally called only by the Paint method in this node’s parent.

setUpMapping

abstract public void setUpMapping()

Description

Initializes the mappings between user and coordinate space.

Remarks

This must be called whenever the screen size, the window or the viewport may have changed. Generally, it is safest to call this each time the chart is repainted.

---

AxisXY Class

public class Imsl.Chart2D.AxisXY : Axis

The axes for an x-y chart.

This node is used when the mapping to and from user and device space can be decomposed into an x and a y mapping. This is when the mapping map(userX,userY) = (deviceX,deviceY) can be written as map(userX,userY) = (mapX(userX), mapY(userY)) = (deviceX,deviceY).

Properties

AxisX

virtual public Imsl.Chart2D.Axis1D AxisX {get; }

Description

The X axis associated with this node.

Property Value

An Axis1D which contains the “AxisX” attribute value.

Remarks

The X axis is a child of this node.

AxisY

virtual public Imsl.Chart2D.Axis1D AxisY {get; }
Description
The Y axis associated with this node.

Property Value
An Axis1D which contains the “AxisY” attribute value.

Remarks
The Y axis is a child of this node.

Constructor

AxisXY
public AxisXY(Imsl.Chart2D.Chart chart)

Description
Creates an AxisXY.

Parameter
chart – A Chart which is the parent of this node.

Remarks
This also creates two Axis1D nodes as children of this node. They hold the decomposed mapping. The “Viewport” attribute for this node is set to [0.2,0.8] by [0.2,0.8].

Methods

GetCross
virtual public double[] GetCross()

Description
Returns the “Cross” attribute value.

Returns

Remarks
The value is the point where the X and Y axes intersect, (xcross,ycross). If “Cross” is not defined then null is returned.

MapDeviceToUser
override public void MapDeviceToUser(int devX, int devY, double[] userXY)

Description
Maps the device coordinates to user coordinates.
Parameters

- devX – An int which specifies the device x-coordinate.
- devY – An int which specifies the device y-coordinate.
- userXY – An int[2] array on input, on output, the user coordinates.

MapUserToDevice

override public void MapUserToDevice(double userX, double userY, int[] devXY)

Description
Maps the user coordinates (userX, userY) to the device coordinates devXY.

Parameters

- userX – A double which specifies the user x-coordinate.
- userY – A double which specifies the user y-coordinate.
- devXY – An int[2] array on input, on output, the device coordinates.

Paint

override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter

- draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetCross

virtual public void SetCross(double xcross, double ycross)

Description
Sets the “Cross” attribute value.

Parameters

- xcross – A double which specifies the x-coordinate where the axes cross.
- ycross – A double which specifies the y-coordinate where the axes cross.

Remarks
This defines the point where the X and Y axes intersect. If “Cross” is not defined then the attribute “Window” is used to determine the crossing point.
Description
Sets the “Cross” attribute value.

Parameter
cross – A double[2] containing the x and y-coordinate where the axes cross.

Remarks
This defines the point where the X and Y axes intersect. If “Cross” is not defined then the attribute “Window” is used to determine the crossing point.

SetUpMapping
override public void SetUpMapping()

Description
Initializes the mappings between user and coordinate space.

Remarks
This must be called whenever the screen size, the window or the viewport may have changed. Generally, it is safest to call this each time the chart is repainted.

SetWindow
virtual public void SetWindow(double[] window)

Description
Sets the window for an AxisXY.

Parameter

Example: Area Chart
An area chart is constructed in this example. Three data sets are used and a legend is added to the chart.

using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class AreaEx1 : FrameChart
{
    public AreaEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
        double dx = .5 * System.Math.PI / (npoints - 1);
        double[] x = new double[npoints];
        double[] y1 = new double[npoints];
        double[] y2 = new double[npoints];
        double[] y3 = new double[npoints];
    
}
// Generate some data
for (int i = 0; i < npoints; i++)
{
    x[i] = i * dx;
    y1[i] = System.Math.Sin(x[i]);
    y2[i] = System.Math.Cos(x[i]);
    y3[i] = System.Math.Atan(x[i]);
}
Data d1 = new Data(axis, x, y1);
Data d2 = new Data(axis, x, y2);
Data d3 = new Data(axis, x, y3);

// Set Data Type to Fill Area
axis.DataType = Imsl.Chart2D.Data.DATA_TYPE_FILL;

// Set Line Colors

// Set Fill Colors

// Set Data Labels
d1.SetTitle("Sine");
d2.SetTitle("Cosine");
d3.SetTitle("ArcTangent");

// Add a Legend
Legend legend = chart.Legend;
legend.SetTitle(new Text("Legend"));
legend.IsVisible = true;

// Set the Chart Title
chart.ChartTitle.SetTitle("Area Plots");
}

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new AreaEx1());
}
Axis1D Class

public class Imsl.Chart2D.Axis1D : ChartNode

An x-axis or a y-axis.

Axis1D is created by Imsl.Chart2D.AxisXY (p. 1348) as its child. It can be retrieved using the method Imsl.Chart2D.AxisXY.AxisX (p. 1348) or Imsl.Chart2D.AxisXY.AxisY (p. 1348).

It in turn creates the following child nodes: Imsl.Chart2D.Axis1D.AxisLine (p. 1354),

Legend
- Sine
- Cosine
- ArcTangent
The number of tick marks ("Number" attribute) is set to 5, but autoscaling can change this value.

**Properties**

**AxisLabel**

```csharp
virtual public Imsl.Chart2D.AxisLabel AxisLabel {get; }
```

**Description**
The label node of this `Axis1D`.

**Property Value**
An `AxisLabel` which contains the "AxisLabel" attribute value.

**Remarks**
This is a child of the axis node.

**AxisLine**

```csharp
virtual public Imsl.Chart2D.AxisLine AxisLine {get; }
```

**Description**
The line node of this `Axis1D`.

**Property Value**
An `AxisLine` that is the axis line node associated with this axis.

**Remarks**
This is a child of the axis node.

**AxisTitle**

```csharp
virtual public Imsl.Chart2D.AxisTitle AxisTitle {get; }
```

**Description**
The title node of this `Axis1D`.

**Property Value**
An `AxisTitle` that is the title node associated with this axis.

**Remarks**
This is a child of the axis node.

**AxisUnit**

```csharp
virtual public Imsl.Chart2D.AxisUnit AxisUnit {get; }
```
Description
The unit node of this Axis1D.

Property Value
An AxisUnit that is the unit node associated with this axis.

Remarks
This is a child of the axis node.

FirstTick
virtual public double FirstTick {get; set; }

Description
This indicates the location of the first tick.

Property Value
A double which contains the “FirstTick” attribute value.

Remarks
By default, FirstTick = GetWindow()[0].

Grid
virtual public Imsl.Chart2D.Grid Grid {get; }

Description
The grid node of this Axis1D.

Property Value
A Grid which contains the “Grid” attribute value.

Remarks
This is a child of the axis node.

MajorTick
virtual public Imsl.Chart2D.MajorTick MajorTick {get; }

Description
The major tick node of this Axis1D.

Property Value
A MajorTick which contains the “MajorTick” attribute value.

Remarks
This is a child of the axis node.

MinorTick
virtual public Imsl.Chart2D.MinorTick MinorTick {get; }

Description
The minor tick node of this Axis1D.
**Property Value**
A MajorTick which contains the “MinorTick” attribute value.

**Remarks**
This is a child of the axis node.

**TickInterval**

```csharp
virtual public double TickInterval {get; set; }
```

**Description**
The tick interval node of this Axis1D.

**Property Value**
A double which contains the “TickInterval” attribute value.

**Type**

```csharp
virtual public int Type {get; set; }
```

**Description**
Specifies the type of this Axis1D.

**Property Value**
An int which contains the type of this axis node.

**Remarks**
The node types are:

- Imsl.Chart2D.AbstractChartNode.AXIS_X (p. 1295)
- Imsl.Chart2D.AbstractChartNode.AXIS_Y (p. 1296)
- Imsl.Chart2D.ChartNode.AXIS_X_TOP (p. 1311)
- Imsl.Chart2D.ChartNode.AXIS_Y_RIGHT (p. 1311)

**Methods**

**GetTicks**

```csharp
virtual public double[] GetTicks()
```

**Description**
Returns the “Ticks” attribute value.

**Returns**
A double[] containing the “Ticks” attribute value.
Remarks
If not set, then computed tick values are returned based on the type of axis (linear, log or custom), and the attributes “Number” and “TickInterval”.

GetWindow
virtual public double[] GetWindow()

Description
Returns the window for an Axis1D.

Returns
A double[2] containing the range of this axis.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetTicks
virtual public void SetTicks(double[] ticks)

Description
Sets the “Ticks” attribute value.

Parameters
ticks – A double[] which contains the location, in user coordinates, of the major tick marks.

SetWindow
virtual public void SetWindow(double min, double max)

Description
Sets the window for an Axis1D.

Parameters
min – A double which specifies the value of the left/bottom end of the axis.
max – A double which specifies the value of the right/top end of the axis.

SetWindow
virtual public void SetWindow(double[] window)
Description
Sets the window for an Axis1D.

Parameter
window – A double[2] containing the range of this axis.

AxisLabel Class

public class Imsl.Chart2D.AxisLabel : ChartNode

The labels on an axis.

AxisLabel is created by Imsl.Chart2D.Axis1D (p. 1353) as its child. It can be retrieved using the method Imsl.Chart2D.Axis1D.AxisLabel (p. 1354).

Axis labels are placed at the tick mark locations. The number of tick marks is determined by the attribute “Number”. Tick marks are evenly spaced. If the attribute “Labels” is defined then it is used to label the tick marks.

If “Labels” is not defined, the ticks are labeled numerically. The endpoint label values are obtained from the attribute “Window”. The numbers are formatted using the attribute “TextFormat”.

Text attributes (specified with CultureInfo, NumberFormatInfo.CurrentInfo and DateTimeFormatInfo.CurrentInfo members) in this node control the drawing of the axis labels.

Methods

GetLabels

virtual public Imsl.Chart2D.Text[] GetLabels()

Description
Returns the “Labels” attribute.

Returns
A Text[] containing the axis labels.

Remarks
By default, GetLabels = null.

Paint

override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.
**Parameter**

`draw` – A `Draw` containing the object to be painted.

**Remarks**

This is normally called only by the `Paint` method in this node’s parent.

**SetLabels**

`virtual public void SetLabels(string[] value)`

**Description**

Sets the axis label values for this node to be used instead of the default numbers.

**Parameter**

`value` – A `String[]` specifying the labels for the major tick marks.

**Remarks**

The attribute “Number” is also set to `value.Length`.

---

**AxisLine Class**

```csharp
public class Imsl.Chart2D.AxisLine : ChartNode

The axis line.

AxisLine is created by Imsl.Chart2D.Axis1D (p. 1353) as its child. It can be retrieved using the method Imsl.Chart2D.Axis1D.AxisLine (p. 1354).

Line attributes (specified with LineColor (p. 1302), LineWidth (p. 1303) and SetMarkerDashPattern (p. 1331)) in this node control the drawing of the axis line.

**Method**

**Paint**

`override public void Paint(Imsl.Chart2D.Draw draw)`

**Description**

Paints this node and all of its children.

**Parameter**

`draw` – A `Draw` with the object to be painted.
Remarks
This is normally called only by the Paint method in this node’s parent.

AxisTitle Class

```csharp
public class Imsl.Chart2D.AxisTitle : ChartNode
```

The title on an axis.

AxisTitle is created by Imsl.Chart2D.Axis1D (p. 1353) as its child. It can be retrieved using the method Imsl.Chart2D.Axis1D.AxisTitle (p. 1354).

The axis title is the “Title” attribute value at this node. Text attributes (specified with CultureInfo, NumberFormatInfo.CurrentInfo and DateTimeFormatInfo.CurrentInfo members) in this node control the drawing of the axis title.

Method

```
Paint
override public void Paint(Imsl.Chart2D.Draw draw)
```

Description
Paints this node and all of its children.

Parameter
- `draw` – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

AxisUnit Class

```csharp
public class Imsl.Chart2D.AxisUnit : ChartNode
```

The unit on an axis.

AxisUnit is created by Imsl.Chart2D.Axis1D (p. 1353) as its child. It can be retrieved using the method Imsl.Chart2D.Axis1D.AxisUnit (p. 1354).
The axis title is the “Title” attribute value at this node. Text attributes (specified with CultureInfo, NumberFormatInfo.CurrentInfo and DateTimeFormatInfo.CurrentInfo members) in this node control the drawing of the unit title.

**Method**

**Paint**

```java
override public void Paint(Imsl.Chart2D.Draw draw)
```

**Description**

Paints this node and all of its children.

**Parameter**

- `draw` – A `Draw` which is to be painted.

**Remarks**

This is normally called only by the `Paint` method in this node’s parent.

---

**MajorTick Class**

```java
public class Imsl.Chart2D.MajorTick : ChartNode
```

The major tick marks.

MajorTick is created by `Imsl.Chart2D.Axis1D` (p. 1353) as its child. It can be retrieved using the `Imsl.Chart2D.Axis1D.MajorTick` (p. 1355) property.

Line attributes (specified with `LineColor` (p. 1302), `LineWidth` (p. 1303) and `SetMarkerDashPattern` (p. 1331)) in this node control the drawing of the major tick marks.

---

**Method**

**Paint**

```java
override public void Paint(Imsl.Chart2D.Draw draw)
```

**Description**

Paints this node and all of its children.

**Parameter**

- `draw` – A `Draw` which is to be painted.
Remarks
This is normally called only by the Paint method in this node’s parent.

---

**MinorTick Class**

```csharp
public class Imsl.Chart2D.MinorTick : ChartNode

The minor tick marks.

MinorTick is created by Imsl.Chart2D.Axis1D (p. 1353) as its child. It can be retrieved using the Imsl.Chart2D.Axis1D.MinorTick (p. 1355) property.

Line attributes (specified with LineColor (p. 1302), LineWidth (p. 1303) and SetMarkerDashPattern (p. 1331)) in this node control the drawing of the minor tick marks.
```

---

**Method**

**Paint**

```csharp
override public void Paint(Imsl.Chart2D.Draw draw)
```

**Description**

Paints this node and all of its children.

**Parameter**

```csharp
draw – A Draw which is to be painted.
```

**Remarks**

This is normally called only by the Paint method in this node’s parent.

---

**Transform Interface**

```csharp
public interface Imsl.Chart2D.Transform

Definitions a custom transformation along an axis.

Axis1D has built in support for linear and logarithmic transformations. Additional transformations can be specified by setting the “CustomTransform” attribute in an Axis1D to an Object that implements this interface.

The interface consists of two methods that must be implemented. Each method is the inverse of the other.
```
**Methods**

**MapUnitToUser**

```java
abstract public double MapUnitToUser(double unit)
```

**Description**
Maps points in the interval \([0,1]\) to user coordinates.

**Parameter**

- **unit** – A `double` which contains a location in unit coordinates to be converted to user coordinates.

**MapUserToUnit**

```java
abstract public double MapUserToUnit(double user)
```

**Description**
Maps user coordinates to the interval \([0,1]\).

**Parameter**

- **user** – A `double` which contains a location in user coordinates to be converted to unit coordinates.

**Remarks**
The user coordinate interval is specified by the “Window” attribute for the axis with which the transform is associated.

**SetupMapping**

```java
abstract public void SetupMapping(Imsl.Chart2D.Axis1D axis1d)
```

**Description**
Initializes the mappings between user and coordinate space.

**Parameter**

- **axis1d** – An `Axis1D` that specifies the axis to which the transform is to be associated.

---

**TransformDate Class**

```java
public class Imsl.Chart2D.TransformDate : Imsl.Chart2D.Transform
```

Defines a transformation along an axis that skips weekend dates.

**Constructor**

```java
TransformDate()
```

---

`Chart2D`
Description
Initializes a new instance of the Imsl.Chart2D.TransformDate (p. 1363) class.

Methods

IsWeekday
virtual public bool IsWeekday(System.DateTime dateTime)

Description
Indicates whether the specified date is a weekday.

Parameter
dateTime – A DateTime indicating the day to be confirmed a day other than Saturday or Sunday.

Returns
A bool indicating whether this is neither Saturday nor Sunday.

Remarks
Returns false if the specified day is a Saturday or Sunday.

MapUnitToUser
virtual public double MapUnitToUser(double unit)

Description
Maps points in the interval [0,1] to user coordinates.

Parameter
unit – A double which contains a location in unit coordinates to be converted to user coordinates.

MapUserToUnit
virtual public double MapUserToUnit(double user)

Description
Maps user coordinates to the interval [0,1].

Parameter
user – A double which contains a location in user coordinates to be converted to unit coordinates.

Remarks
The user coordinate interval is specified by the “Window” attribute for the axis with which the transform is associated.

SetupMapping
virtual public void SetupMapping(Imsl.Chart2D.Axis1D axis1d)
**Description**
Initializes the mappings between user and coordinate space.

**Parameter**
- axis1d – An Axis1D that specifies the axis to which the transform is to be associated.

---

**AxisR Class**

```java
public class Imsl.Chart2D.AxisR : ChartNode
```

The R-axis in a polar plot.

AxisR is created by `Imsl.Chart2D.Polar` (p. 1495) as its child. It can be retrieved using the `Imsl.Chart2D.Polar.AxisR` (p. 1495).

It in turn creates the following child nodes: `Imsl.Chart2D.AxisR.AxisRLine` (p. 1365), `Imsl.Chart2D.AxisR.AxisRLabel` (p. 1365) and `Imsl.Chart2D.AxisR.AxisRMajorTick` (p. 1366).

The number of tick marks (“Number” attribute) is set to 4, but autoscaling can change this value.

---

**See Also**

`Imsl.Chart2D.Polar` (p. 1495)

---

**Properties**

---

**AxisRLabel**

```java
virtual public Imsl.Chart2D.AxisRLabel AxisRLabel {get; }
```

**Description**
A `AxisRLabel` which specifies the label node associated with this axis.

---

**AxisRLine**

```java
virtual public Imsl.Chart2D.AxisRLine AxisRLine {get; }
```

**Description**
Specifies the line node associated with this axis.
**Property Value**
A AxisRLine which contains the line node associated with this axis.

**AxisRMajorTick**
virtual public Imsl.Chart2D.AxisRMajorTick AxisRMajorTick {get; }

**Description**
Specifies the major tick associated with this axis.

**Property Value**
A AxisRMajorTick which contains the “AxisRMajorTick” attribute value.

**Remarks**
This is a child of the axis node.

**TickInterval**
virtual public double TickInterval {get; set; }

**Description**
The tick interval node of this AxisR.

**Property Value**
A double which contains the “TickInterval” attribute value.

**Window**
virtual public double Window {get; set; }

**Description**
The radius at which AxisTheta is drawn.

**Property Value**
A double which contains the “Window” attribute value.

**Remarks**
The window has a maximum value of R. The R-axis always starts at 0. By default, window = 1.0.

**Methods**

**GetTicks**
virtual public double[] GetTicks()

**Description**
Returns the “Ticks” attribute value.

**Returns**
A double[] containing the “Ticks” attribute value.
Remarks
If not set, then computed tick values are returned based on the type of axis (linear, log or custom), the attributes “Number” and “TickInterval”.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

AxisRLabel Class

public class Imsl.Chart2D.AxisRLabel : ChartNode
The labels on an axis.

AxisRLabel is created by Imsl.Chart2D.AxisR (p. 1365) as its child. It can be retrieved using the method Imsl.Chart2D.AxisR.AxisRLabel (p. 1365).

Axis labels are placed at the tick mark locations. The number of tick marks is determined by the attribute “Number”. Tick marks are evenly spaced. If the attribute “Labels” is defined then it is used to label the tick marks.

If “Labels” is not defined, the ticks are labeled numerically. The endpoint label values are obtained from the attribute “Window”. The numbers are formatted using the attribute “TextFormat”.

Text attributes (specified with CultureInfo, NumberFormatInfo.CurrentInfo and DateTimeFormatInfo.CurrentInfo members) in this node control the drawing of the axis labels.

See Also
Imsl.Chart2D.Polar (p. 1495), Imsl.Chart2D.AxisR (p. 1365)

Methods

GetLabels
virtual public Imsl.Chart2D.Text[] GetLabels()
Description
Returns the “Labels” attribute.

Returns
A Text[] containing the axis labels.

Remarks
By default, GetLabels = null.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetLabels
virtual public void SetLabels(string[] value)

Description
Sets the axis label values for this node to be used instead of the default numbers.

Parameter
value – A String[] specifying the labels for the major tick marks.

Remarks
The attribute “Number” is also set to value.Length.

AxisRLine Class

public class Imsl.Chart2D.AxisRLine : ChartNode
The radius axis line in a polar plot.

AxisRLine is created by Imsl.Chart2D.AxisR (p. 1365) as its child. It can be retrieved using the method Imsl.Chart2D.AxisR.AxisRLine (p. 1365).

Line attributes (specified with LineColor (p. 1302), LineWidth (p. 1303) and SetMarkerDashPattern (p. 1331)) in this node control the drawing of the axis line.
See Also

Imsl.Chart2D.Polar (p. 1495), Imsl.Chart2D.AxisR (p. 1365)

Method

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

AxisRMajorTick Class

public class Imsl.Chart2D.AxisRMajorTick : ChartNode
The major tick marks for the radius axis in a polar plot.
AxisRMajorTick is created by Imsl.Chart2D.AxisR (p. 1365) as its child. It can be retrieved using the method Imsl.Chart2D.AxisR.AxisRMajorTick (p. 1366).

Line attributes (specified with LineColor (p. 1302), LineWidth (p. 1303) and SetMarkerDashPattern (p. 1331)) in this node control the drawing of the major tick marks.

See Also

Imsl.Chart2D.Polar (p. 1495), Imsl.Chart2D.AxisR (p. 1365)

Method

Paint
override public void Paint(Imsl.Chart2D.Draw draw)
**Description**
Paints this node and all of its children.

**Parameter**
- **draw** – A Draw which is to be painted.

**Remarks**
This is normally called only by the Paint method in this node’s parent.

---

### AxisTheta Class

```csharp
class Imsl.Chart2D.AxisTheta : ChartNode
```

The angular axis in a polar plot.

*AxisTheta* is created by *Imsl.Chart2D.Polar* (p. 1495) as its child. It can be retrieved using the *Imsl.Chart2D.Polar.AxisTheta* (p. 1495) property.

The angles are labeled using the “TextFormat” attribute, which is set to ‘‘0. ##\u00b0’’, where \u00b0 is the Unicode character for degrees. This labels the angles in degrees. More generally, “TextFormat” can be set to a *NumberFormat* object to format the angles in degrees.

“TextFormat” can also be set to a *MessageFormat* object. In this case, field {0} is the value in degrees, field {1} is the value in radians and field {2} is the value in radians/π. So, for labels like 1.5\u03c0, where \u03c0 is the Unicode character for π, set “TextFormat” to new *MessageFormat*('‘{2,number,0.##\u03c0}’’).

The number of tick marks (“Number” attribute) is set to 9, but autoscaling can change this value.

---

**See Also**

*Imsl.Chart2D.Polar* (p. 1495)

---

**Methods**

**GetTicks**

```csharp
virtual public double[] GetTicks()
```

**Description**
Returns the “Ticks” attribute value.
Returns
A double[] containing the “Ticks” attribute value.
Remarks
These are the positions at which the angles are labeled. The ticks are in radians, not degrees.

GetWindow
virtual public double[] GetWindow()

Description
Returns the window for an AxisTheta.
Returns
A double array of length two containing the angular range of the window.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.
Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetWindow
virtual public void SetWindow(double min, double max)

Description
Sets the window for an AxisTheta.
Parameters
min – A double which specifies the initial angular value, in radians value.
max – A double which specifies the final angular value, in radians.

Remarks
The default “Window” is [0,2pi].

SetWindow
virtual public void SetWindow(double[] window)

Description
Sets the window for an AxisTheta.
Parameter
window – A double array of length two containing the angular range.
Remarks
The default “Window” is [0,2\pi].

GridPolar Class

public class Imsl.Chart2D.GridPolar : ChartNode

Draws the grid lines for a polar plot.

GridPolar is created by Imsl.Chart2D.Polar (p. 1495) as its child. It can be retrieved using the Imsl.Chart2D.Polar.GridPolar (p. 1495) property.

Line attributes (specified with LineColor (p. 1302), LineWidth (p. 1303) and SetMarkerDashPattern (p. 1331)) in this node control the drawing of the grid lines.

Method

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter

draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

Data Class

public class Imsl.Chart2D.Data : ChartNode

A data node in the chart tree.

Drawing of a Data node is determined by the DataType (p. 1320) property. Multiple bits can be set in “DataType”.

If the DATA_TYPE_LINE (p. 1312) bit is set, the line attributes are active.
If the DATA_TYPE_MARKER (p. 1312) bit is set, the marker attributes are active.
If the DATA_TYPE_FILL (p. 1312) bit is set, the fill attributes are active.
If LabelType (p. 1302) is set to something other than the default (LABEL_TYPE_NONE), then the data points are labeled. The contents of the labels are determined by the value of the LabelType property.

The drawing of the labels is controlled with CultureInfo, NumberFormatInfo.CurrentInfo and DateTimeFormatInfo.CurrentInfo members) in this node control the drawing of the title.

**Constructors**

**Data**

```csharp
public Data(Imsl.Chart2D.ChartNode parent)
```

**Description**

Creates a data node.

**Parameter**

parent – A ChartNode which specifies the parent of this data node.

```csharp
public Data(Imsl.Chart2D.ChartNode parent, double[] y)
```

**Description**

Creates a Data node with y values.

**Parameters**

parent – A ChartNode which specifies the parent of this data node.

y – A double array containing the dependant values for this node.

**Remarks**

The x values are set to the double array containing \{0,1,\ldots,y.Length-1\}.

```csharp
public Data(Imsl.Chart2D.ChartNode parent, Imsl.Chart2D.ChartFunction cf, double a, double b)
```

**Description**

Creates a Data node with y values.

**Parameters**

parent – A ChartNode which specifies the parent of this data node.

cf – A ChartFunction that defines the function to be plotted.

a – A double that contains the left endpoint.

b – A double that contains the right endpoint.
Remarks
The x values are set to the double array containing \{0,1,\ldots,y.Length-1\}.

Data
public Data(Imsl.Chart2D.ChartNode parent, double[] x, double[] y)

Description
Creates a Data node with x and y values.

Parameters
  parent – A ChartNode which specifies the parent of this data node.
  x – A double array containing the independant values for this node.
  y – A double array containing the dependant values for this node.

Methods

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
  draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetDataRange
virtual public void SetDataRange(double[] range)

Description
Update the data range, range = \{xmin,xmax,ymin,ymax\}

Parameter
  range – a double array which contains the updated range, \{xmin,xmax,ymin,ymax\}

Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.
Example: Scatter Chart

A scatter plot is constructed in this example. Three data sets are used and a legend is added to the chart.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class ScatterEx1 : FrameChart
{
    public ScatterEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
        double dx = .5 * System.Math.PI / (npoints - 1);
        double[] x = new double[npoints];
        double[] y1 = new double[npoints];
        double[] y2 = new double[npoints];
        double[] y3 = new double[npoints];

        // Generate some data
        for (int i = 0; i < npoints; i++)
        {
            x[i] = i * dx;
            y1[i] = System.Math.Sin(x[i]);
            y2[i] = System.Math.Cos(x[i]);
            y3[i] = System.Math.Atan(x[i]);
        }
        Data d1 = new Data(axis, x, y1);
        Data d2 = new Data(axis, x, y2);
        Data d3 = new Data(axis, x, y3);

        // Set Data Type to Marker
        d1.DataType = Imsl.Chart2D.Data.DATA_TYPE_MARKER;
        d2.DataType = Imsl.Chart2D.Data.DATA_TYPE_MARKER;
        d3.DataType = Imsl.Chart2D.Data.DATA_TYPE_MARKER;

        // Set Marker Types
        d1.MarkerType = Data.MARKER_TYPE_CIRCLE_PLUS;
        d2.MarkerType = Data.MARKER_TYPE_HOLLOW_SQUARE;
        d3.MarkerType = Data.MARKER_TYPE_ASTERISK;

        // Set Marker Colors

        // Set Data Labels
        d1.SetTitle("Sine");
        d2.SetTitle("Cosine");
        d3.SetTitle("ArcTangent");

        // Add a Legend
        Legend legend = chart.Legend;
    }
}
```
legend.SetTitle(new Text("Legend"));
legend.IsVisible = true;

// Set the Chart Title
chart.ChartTitle.SetTitle("Scatter Plot");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new ScatterEx1());
}

Output
Example: Line Chart

A simple line chart is constructed in this example. Three data sets are used and a legend is added to the chart. An annotation is included indicating a feature on the chart.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class LineEx1 : FrameChart
{
    public LineEx1()
    {
        Chart chart = this.Chart;

        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
        double dx = .5 * System.Math.PI / (npoints - 1);
        double[] x = new double[npoints];
        double[] y1 = new double[npoints];
        double[] y2 = new double[npoints];
        double[] y3 = new double[npoints];

        // Generate some data
        for (int i = 0; i < npoints; i++)
        {
            x[i] = i * dx;
            y1[i] = System.Math.Sin(x[i]);
            y2[i] = System.Math.Cos(x[i]);
            y3[i] = System.Math.Atan(x[i]);
        }
        Data d1 = new Data(axis, x, y1);
        Data d2 = new Data(axis, x, y2);
        Data d3 = new Data(axis, x, y3);

        // Set Data Type to Line
        axis.DataType = Imsl.Chart2D.AxisXY.DATA_TYPE_LINE;

        // Set Line Colors

        // Set Data Labels
        d1.SetTitle("Sine");
        d2.SetTitle("Cosine");
        d3.SetTitle("ArcTangent");

        // Add a Legend
        Legend legend = chart.Legend;
        legend.SetTitle(new Text("Legend"));
        legend.IsVisible = true;
    }
}
```
// Add an Annotation
Text label = new Text("Maximum Values");
label.Alignment = Axis.TEXT_X_RIGHT;
Annotation ann = new Annotation(axis, label, 1.6, 1.02);

// Set the Chart Title
chart.ChartTitle.SetTitle("Line Plots");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new LineEx1());
}

Example: Picture Chart

A picture plot is constructed in this example.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;
using System.Drawing;

public class PictureEx1 : FrameChart
{
    public PictureEx1()
    {
        string appPath = Application.ExecutablePath;
    }
}
```
Chart chart = this.Chart;
AxisXY axis = new AxisXY(chart);

int npoints = 20;
double dx = .5 * System.Math.PI / (npoints - 1);
double[] x = new double[npoints];
double[] y1 = new double[npoints];
double[] y2 = new double[npoints];

// Generate some data
for (int i = 0; i < npoints; i++)
{
    x[i] = i * dx;
    y1[i] = System.Math.Sin(x[i]);
    y2[i] = System.Math.Cos(x[i]);
}
Data d1 = new Data(axis, x, y1);
Data d2 = new Data(axis, x, y2);

// Load Images
d1.DataType = Data.DATA_TYPE_PICTURE;
d1.ImageAttr = new Bitmap(@"IMSL.NET\Example\Chart2D\marker.gif", true);
d2.DataType = Data.DATA_TYPE_PICTURE;
d2.ImageAttr = new Bitmap(@"IMSL.NET\Example\Chart2D\marker2.gif", true);

// Set the Chart Title
chart.ChartTitle.SetTitle("Picture Plot");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new PictureEx1());
}
}
Example: Area Chart

An area chart is constructed in this example. Three data sets are used and a legend is added to the chart.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class AreaEx1 : FrameChart
{
    public AreaEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
    }
}```
double dx = .5 * System.Math.PI / (npoints - 1);
double[] x = new double[npoints];
double[] y1 = new double[npoints];
double[] y2 = new double[npoints];
double[] y3 = new double[npoints];

    // Generate some data
    for (int i = 0; i < npoints; i++)
    {
        x[i] = i * dx;
        y1[i] = System.Math.Sin(x[i]);
        y2[i] = System.Math.Cos(x[i]);
        y3[i] = System.Math.Atan(x[i]);
    }
    Data d1 = new Data(axis, x, y1);
    Data d2 = new Data(axis, x, y2);
    Data d3 = new Data(axis, x, y3);

    // Set Data Type to Fill Area
    axis.DataType = Imsl.Chart2D.Data.DATA_TYPE_FILL;

    // Set Line Colors

    // Set Fill Colors

    // Set Data Labels
    d1.SetTitle("Sine");
    d2.SetTitle("Cosine");
    d3.SetTitle("ArcTangent");

    // Add a Legend
    Legend legend = chart.Legend;
    legend.SetTitle(new Text("Legend");
    legend.IsVisible = true;

    // Set the Chart Title
    chart.ChartTitle.SetTitle("Area Plots");
}

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new AreaEx1());
}
ChartFunction Interface

public interface Imsl.Chart2D.ChartFunction
An interface that allows a function to be plotted.
See Also

Imsl.Chart2D.Data (p. 1372)

Method

F
abstract public double F(double x)

Description
Function to be charted.

Parameter
x – A double[] which specifies the independent data.

Returns
A double[] containing the dependant data.

ChartSpline Class

public class Imsl.Chart2D.ChartSpline : Imsl.Chart2D.ChartFunction
Wraps a spline into a ChartFunction to be plotted.

Constructors

ChartSpline
public ChartSpline(Imsl.Math.Spline spline)

Description
Creates a ChartSpline.

Parameter
spline – A Spline used to construct this ChartSpline.

ChartSpline
public ChartSpline(Imsl.Math.Spline spline, int ideriv)
Description
Creates a ChartSpline.

Parameters
spline – A Spline which is to have its derivative plotted.
ideriv – An int that specifies what derivative is to be plotted.

Remarks
If zero, the function value is plotted.
If one, the first derivative is plotted, etc.

Method

F
virtual public double F(double x)

Description
Function to be charted.

Parameter
x – A double specifying the point at which the function is to be evaluated.

Returns
A double containing the function evaluation.

Text Class

public class Imsl.Chart2D.Text
The value of the attribute “Title”.
A title is a multi-line string with alignment information.
Line breaks are indicated by the newline character (“n”) within the string.
Titles are drawn relative to a reference point. Alignment determines the position of the reference point on the horizontally-aligned box that bounds the text.
Properties

Alignment
virtual public int Alignment {get; set; }

Description
The alignment for this Text object.

Property Value
An int indicating the alignment for this Text object.

Remarks
The alignment determines the position of the reference point on the horizontally aligned box containing the drawn text. It is the bitwise combination of the following:

- TEXT_X_LEFT (p. 1317)
- TEXT_X_CENTER (p. 1317)
- TEXT_X_RIGHT (p. 1317)
- TEXT_Y_BOTTOM (p. 1317)
- TEXT_Y_CENTER (p. 1318)
- TEXT_Y_TOP (p. 1318)

DefaultAlignment
virtual public int DefaultAlignment {set; }

Description
The default alignment for this Text object.

Property Value
An int indicating the alignment for this Text object.

Remarks
The alignment determines the position of the reference point on the horizontally aligned box containing the drawn text. It is the bitwise combination of the following:

- TEXT_X_LEFT (p. 1317)
- TEXT_X_CENTER (p. 1317)
- TEXT_X_RIGHT (p. 1317)
- TEXT_Y_BOTTOM (p. 1317)
- TEXT_Y_CENTER (p. 1318)
- TEXT_Y_TOP (p. 1318)

DefaultOffset
virtual public double DefaultOffset {set; }
Description
The default value of the offset.

Property Value
A double which contains the default offset for this Text object.

Remarks
Offset is in units of the default marker size. Text drawn is offset in the direction of the alignment.

Offset
virtual public double Offset {get; set; }

Description
The offset for this Text object.

Property Value
A double which contains the offset for this Text object.

Remarks
Offset is in units of the default marker size. Text drawn is offset in the direction of the alignment.

String
virtual public string String {get; set; }

Description
A string representation of this Text object.

Property Value
A string containing the value of this Text object.

Constructors

Text
public Text(string text)

Description
Constructs a Text object from a string.

Parameter
text – A string that is to be converted to a Text object.

Text
public Text(string text, int alignment)

Description
Constructs a Text object from a string with specified alignment.
Parameters

- **text** – The String that is to be converted to a Text object.
- **alignment** – An int which specifies the alignment.

Remarks

The alignment determines the position of the reference point on the horizontally aligned box containing the drawn text. It is the bitwise combination of the following:

- TEXT_X_LEFT (p. 1317)
- TEXT_X_CENTER (p. 1317)
- TEXT_X_RIGHT (p. 1317)
- TEXT_Y_BOTTOM (p. 1317)
- TEXT_Y_CENTER (p. 1318)
- TEXT_Y_TOP (p. 1318)

Text

```csharp
public Text(string format, System.IFormatProvider formatProvider,
System.IFormattable obj)
```

Description

Constructs a Text object given a format string, an IFormatProvider and the value to be formatted.

Parameters

- **format** – A string containing the format.
- **formatProvider** – An IFormatProvider like NumberFormat or DateTimeFormat.
- **obj** – A IFormattable that is to be converted into a Text object.

ToolTip Class

```csharp
public class Imsl.Chart2D.ToolTip : ChartNode
```

A tool tip for a chart element.

This class requires that the chart’s component be a subclass of ComponentModel. The ComponentModel class can be subclassed to provide different behaviors for displaying tool tips.

To use, create an instance of ToolTip to activate the ToolTips in a node and in the node’s descendants. The ToolTip string is the value of a node’s “ToolTip” attribute or, if it is null, the node’s “Title” attribute.
Constructor

**ToolTip**

```java
public ToolTip(Imsl.Chart2D.ChartNode parent)
```

**Description**

Creates a ToolTip node that enables tool tips on charts.

**Parameter**

- `parent` – The `ChartNode` parent of this node.

**Remarks**

Do not use the root `ChartNode` for this argument, because it will normally select only the `Background` node.

Methods

**MouseMove**

```java
virtual public void MouseMoved(object sender, System.Windows.Forms MouseEventArgs e)
```

**Description**

The `MouseMove` delegate added to the Chart when a ToolTip is created.

**Parameters**

- `sender` – A `Object` that specifies the sender of an event.
- `e` – A `MouseEventArgs` that provides data for the MouseUp, MouseDown, and MouseMove events.

**Paint**

```java
override public void Paint(Imsl.Chart2D.Draw draw)
```

**Description**

Paints this node and all of its children.

**Parameter**

- `draw` – A `Draw` which is to be painted.

**Remarks**

This is normally called only by the `Paint` method in this node’s parent.
FillPaint Class

public class Imsl.Chart2D.FillPaint
A collection of methods to create Brush objects for fill areas.
All of the Brush objects returned by the methods in this class are serializable.

Methods

Checkerboard

Description
Returns a checkerboard pattern.

Parameters
n – An int that specifies the pattern size in pixels.
colorA – A Color which specifies the first color in the checkerboard pattern.
colorB – A Color which specifies the second color in the checkerboard pattern.

Returns
A Brush containing the checkerboard pattern.

Crosshatch

Description
Returns a crosshatch pattern.

Parameters
n – An int that specifies the pattern size in pixels.
p – An int which specifies the number of pixels between the crosshatched lines.
colorBackground – A Color which specifies the background color.
colorLine – A Color which specifies the color of the line.
Returns
A Brush containing the pattern.

DefaultReadObject
static public void
DefaultReadObject(System.Runtime.Serialization.SerializationInfo info,

Description
Reads the serialized fields written by the DefaultWriteObject method.

Parameters
- info – A SerializationInfo parameter from the special deserialization constructor.
- context – A StreamingContext parameter from the special deserialization constructor.
- instance – An Object to deserialize.

DefaultWriteObject
static public void
DefaultWriteObject(System.Runtime.Serialization.SerializationInfo info,

Description
Writes the serializable fields to the SerializationInfo object, which stores all the data needed to serialize the specified Object.

Parameters
- info – A SerializationInfo parameter from the GetObjectData method.
- context – A StreamingContext parameter from the GetObjectData method.
- instance – An Object to serialize.

Diagonal
System.Drawing.Color colorB)

Description
Returns a diagonal pattern.

Parameters
- n – An int that specifies the pattern size in pixels.
- colorA – A Color which specifies the first color in the diagonal pattern.
- colorB – A Color which specifies the second color in the diagonal pattern.

Returns
A Brush containing the diagonal pattern.

Diamond
Description
Returns a diamond pattern (a checkerboard rotated 45 degrees).

Parameters
- \( n \) – An int that specifies the pattern size in pixels.
- \( p \) – An int which specifies the line thickness.
- colorBackground – A Color which specifies the background color.
- colorLine – A Color which specifies the color of the line.

Returns
A Brush containing the diamond pattern.

DiamondHatch

Description
Returns a crosshatch pattern on a 45 degree angle.

Parameters
- \( n \) – An int that specifies the pattern size in pixels.
- \( p \) – An int which specifies the number of pixels between the crosshatched lines.
- colorBackground – A Color which specifies the background color.
- colorLine – A Color which specifies the color of the line.

Returns
A Brush containing the pattern.

Dot

Description
Returns a pattern that is an array of circles.

Parameters
- \( n \) – An int that specifies the pattern size in pixels.
- \( r \) – An int which specifies the radius of circles in the pattern in pixels.
- colorBackground – A Color which specifies the background color.
- colorCircle – A Color which specifies the color of circles in the pattern.

Returns
A Brush containing the pattern.

HorizontalStripe
**Description**
Returns a horizontally striped pattern.

**Parameters**
- `n` – An `int` that specifies the pattern size in pixels.
- `p` – An `int` which specifies the number of pixels between horizontally lines.
- `colorBackground` – A `Color` which specifies the background color.
- `colorLine` – A `Color` which specifies the color of the line.

**Returns**
A `Brush` containing the pattern.

---

**Image**

**Description**
Returns a tiling of an image.

**Parameter**
- `imageIcon` – An `Image` that specifies the image to be tiled.

**Returns**
A `Brush` containing the tiling of the image.

---

**VerticalStripe**

**Description**
Returns a vertically striped pattern.

**Parameters**
- `n` – An `int` that specifies the pattern size in pixels.
- `p` – An `int` which specifies the number of pixels between vertical lines.
- `colorBackground` – A `Color` which specifies the background color.
- `colorLine` – A `Color` which specifies the color of the line.

**Returns**
A `Brush` containing the pattern.

---

**Draw Class**

```java
public class Imsl.Chart2D.Draw
Renders the chart tree to the screen.
```
Properties

**ClipBounds**

```csharp
virtual public System.Drawing.Rectangle ClipBounds {get; set; }
```

**Description**
Contains the rectangle to be used for clipping.

**Property Value**
A `Rectangle` object which contains the clipping bounds.

**DeviceMarkerSize**

```csharp
virtual public float DeviceMarkerSize {get; }
```

**Description**
The marker size in device coordinates.

**Property Value**
A `float` that contains the marker size.

**Node**

```csharp
virtual public Imsl.Chart2D.ChartNode Node {set; }
```

**Description**
Specifies a `ChartNode` as the current node.

**Property Value**
A `ChartNode` containing the current node.

**Remarks**
This is used to get drawing attributes from the tree.

**ScaleFont**

```csharp
virtual public double ScaleFont {get; set; }
```

**Description**
The factor by which fonts are to be scaled.

**Property Value**
A `double` containing the font scaling factor.

Constructor

**Draw**

```csharp
```

1394 • Draw

IMSL C# Numerical Library
Description
Contracts a Draw object.

Parameters
  graphics – A Graphics object encapsulating a GDI+ drawing surface.
  bounds – A Size specifying the width and height of a rectangle.

Methods

CreateGradientBrush
static public System.Drawing.Drawing2D.LinearGradientBrush
CreateGradientBrush(float x1, float y1, System.Drawing.Color color1, float x2,
float y2, System.Drawing.Color color2)

Description
Creates an acyclic GradientBrush.

Parameters
  x1 – A float containing the x-coordinate of the upper-left corner of drawing area.
  y1 – A float containing the y-coordinate of the upper-left corner of drawing area.
  color1 – A Color structure that represents the starting color for the gradient.
  x2 – A float containing the x-coordinate of the lower-right corner of drawing area.
  y2 – A float containing the y-coordinate of the lower-right corner of drawing area.
  color2 – A Color structure that represents the ending color for the gradient.

Returns
A new instance of LinearGradientBrush with the colors and coordinates specified.

Remarks
This gradient is acyclic.

DrawArc
virtual public void DrawArc(int x, int y, int width, int height, int
startAngle, int arcAngle)

Description
Draws the outline of a circular or elliptical arc covering the specified rectangle.

Parameters
  x – An int which contains the x-coordinate of the upper-left corner of the rectangle that defines
the ellipse.
  y – An int which contains the y-coordinate of the upper-left corner of the rectangle that defines
the ellipse.
width – An int which contains the width of the rectangle that defines the ellipse.

height – An int which contains the height of the rectangle that defines the ellipse.

startAngle – An int which specifies the angle in degrees measured clockwise from the x-axis to the starting point of the arc.

arcAngle – An int which specifies an angle in degrees measured counter-clockwise from the startAngle parameter to the ending point of the arc.

Remarks
The center of the arc is center of this rectangle.
startAngle = 0 is equivalent to the 3-o’clock position.

DrawArc draws the arc from startAngle to startAngle + arcAngle. A positive arcAngle indicates a counter-clockwise rotation. A negative arcAngle implies a clockwise rotation.

DrawClippedImage
virtual public void DrawClippedImage(System.Drawing.Image image, int x, int y)

Description
Draw an image with clipping.

Parameters
image – The Image object to draw.
x – An int which specifies the x-coordinate of the upper-left corner of the drawn image.
y – An int which specifies the y-coordinate of the upper-left corner of the drawn image.

DrawErrorBar
virtual public void DrawErrorBar(int x0, int y0, int x1, int y1, int flag)

Description
Draws an error bar.

Parameters
x0 – An int which specifies the x-coordinate of the beginning reference point.
y0 – An int which specifies the y-coordinate of the beginning reference point.
x1 – An int which specifies the x-coordinate of the ending reference point.
y1 – An int which specifies the y-coordinate of the ending reference point.
flag – An int that indicates which caps to draw.

Remarks
Legal values: 0=none, 1=bottom, 2=top, 3=both

DrawImage
virtual public void DrawImage(System.Drawing.Image image, int x, int y)
**Description**
Draws the specified image at the location specified by a coordinate pair.

**Parameters**
- image – The Image object to draw.
- x – An int which specifies the x-coordinate of the upper-left corner of the drawn image.
- y – An int which specifies the x-coordinate of the upper-left corner of the drawn image.

**DrawLine**
virtual public void DrawLine(int x0, int y0, int x1, int y1)

**Description**
Draws a line from between two points.

**Parameters**
- x0 – An int which specifies the x-coordinate of the line origin, (x0,y0).
- y0 – An int which specifies the y-coordinate of the line origin, (x0,y0).
- x1 – An int which specifies the x-coordinate of the line destination, (x1,y1).
- y1 – An int which specifies the y-coordinate of the line destination, (x1,y1).

**DrawMarker**
virtual public void DrawMarker(int x, int y)

**Description**
Draws a marker.

**Parameters**
- x – An int which specifies the x-coordinate of the marker destination, (x,y).
- y – An int which specifies the y-coordinate of the marker destination, (x,y).

**DrawText**
virtual public System.Drawing.Size DrawText(Imsl.Chart2D.Text text, int x, int y)

**Description**
Draws a Text object.

**Parameters**
- text – A Text object to be drawn.
- x – An int which specifies the abscissa of the (x,y) point at which to start drawing the text.
- y – An int which specifies the ordinate of the (x,y) point at which to start drawing the text.

**Returns**
A Size containing the bounds of the Text to be drawn.

**EndErrorBar**
virtual public void EndErrorBar()
Finish drawing an error bar.

**EndFill**

```
virtual public void EndFill()
```

Finish drawing a filled region.

**EndImage**

```
virtual public void EndImage()
```

Finish drawing an image.

**EndLine**

```
virtual public void EndLine()
```

Finish drawing lines.

**EndMarker**

```
virtual public void EndMarker()
```

Finish drawing markers.

**EndText**

```
virtual public void EndText()
```

Finish drawing text.

**FillArc**

```
virtual public void FillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

Fills a circular or elliptical arc covering the specified rectangle.

**Parameters**

- **x** – An `int` which specifies the x-coordinate of the upper-left corner of the rectangular region that defines the ellipse from which the arc is drawn.
- **y** – An `int` which specifies the y-coordinate of the upper-left corner of the rectangular region that defines the ellipse from which the arc is drawn.
- **width** – An `int` which specifies the width of the rectangular region that defines the ellipse from which the arc is drawn.
height – An int which specifies the height of the rectangular region that defines the ellipse from which the arc is drawn.

startAngle – An int which specifies the starting angle of the arc, measured in degrees clockwise from the x-axis.

arcAngle – An int which specifies an angle in degrees measured counter-clockwise from the startAngle parameter to the ending point of the arc.

Remarks
The center of the arc is center of this rectangle.
startAngle = 0 is equivalent to the 3-o’clock position.

DrawArc draws the arc from startAngle to startAngle+arcAngle. A positive arcAngle indicates a counter-clockwise rotation. A negative arcAngle implies a clockwise rotation.

---

**FillPolygon**

virtual public void FillPolygon(System.Drawing.Drawing2D.GraphicsPath polygon)

**Description**
Fill a polygon defined by a Polygon object.

**Parameter**

polygon – A Polygon object which specifies the polygon to be filled.

---

**FillPolygon**

virtual public void FillPolygon(int[] xpoints, int[] ypoints, int npoints)

**Description**
Fills a polygon.

**Parameters**

xpoints – An int array which contains the abscissae of the points which define the polygon.

ypoints – An int array which contains the ordinates of the points which define the polygon.

npoints – An int which specifies the number of points to add to the graphics path.

---

**FillRectangle**

virtual public void FillRectangle(int x, int y, int width, int height)

**Description**
Fill a rectangle.

**Parameters**

x – An int which specifies the x-coordinate of the upper-left corner of the rectangle.

y – An int which specifies the y-coordinate of the upper-left corner of the rectangle.

width – An int which specifies the width of the rectangle.

height – An int which specifies the height of the rectangle.

---

**GetStringWidth**

static public int GetStringWidth(string target, System.Drawing.Font font)
Description
Gets the width of a string.

Parameters
- target – A string to measure.
- font – A Font object that defines the text format of the string.

Returns
An int that represents the size, in pixels, of the string specified by target as drawn with font.

Start
virtual public void Start(Imsl.Chart2D.Chart chart)

Description
Called just before a chart is drawn.

Parameter
- chart – The Chart object to draw.

StartErrorBar
virtual public void StartErrorBar()

Description
Start drawing an ErrorBar.

StartFill
virtual public void StartFill()

Description
Start drawing a filled region.

StartImage
virtual public void StartImage()

Description
Start drawing an image.

StartLine
virtual public void StartLine()

Description
Start drawing a line.

StartMarker
virtual public void StartMarker()
Description
Start drawing a marker.

StartText
virtual public void StartText()

Description
Start drawing text.

Stop
virtual public void Stop()

Description
Called when a chart is finished being drawn.

Translate
virtual public void Translate(int x, int y)

Description
Prepends the specified translation to the transformation matrix of this Graphics object.

Parameters
x – An int which specifies dx, the x component of the translation.
y – An int which specifies dy, the y component of the translation.

FrameChart Class

public class Imsl.Chart2D.FrameChart : Form :
System.ComponentModel.IComponent, System.IDisposable,
System.Windows.Forms.IDropTarget, System.ComponentModel.ISynchronizeInvoke,
System.Windows.Forms.IContainerControl

FrameChart is a Form that contains a chart.

Properties

Chart
virtual public Imsl.Chart2D.Chart Chart {get; set; }
**Description**
Specifies the chart to be handled.

**Property Value**
A Chart that specifies the chart to be displayed by this container.

**Panel**
virtual public Imsl.Chart2D.PanelChart Panel {get; }

**Description**
Specifies a Panel that contains the Chart to be drawn.

**Property Value**
A PanelChart into which the chart is to be drawn.

---

**Constructors**

**FrameChart**
public FrameChart()

**Description**
Creates new FrameChart to display a chart.

**FrameChart**
public FrameChart(Imsl.Chart2D.Chart chart)

**Description**
Creates new FrameChart to display a given chart.

**Parameter**
  chart – A Chart containing the chart to be displayed.

---

**Method**

**Dispose**
override void Dispose(bool disposing)

**Description**
Clean up any resources being used.

**Parameter**
  disposing – A bool indicating whether to release both managed and unmanaged resources.
Remarks
true to release both managed and unmanaged resources; false to release only unmanaged resources.

PanelChart Class

public class Imsl.Chart2D.PanelChart : Panel :
System.ComponentModel.IComponent, System.IDisposable,
System.Windows.Forms.IDropTarget, System.ComponentModel.ISynchronizeInvoke,

A Windows.Forms.Panel that contains a chart.
This class causes the contained chart to be redrawn as necessary.

Properties

Chart
virtual public Imsl.Chart2D.Chart Chart {get; set; }

Description
Specifies the Chart to be rendered for in this panel.

Property Value
A Chart to be handled by this container.

DefaultSize
override System.Drawing.Size DefaultSize {get; }

Description
The default size for a PanelChart is 200x200.

Constructors

PanelChart
public PanelChart()

Description
Creates a new PanelChart.
Remarks
This creates a new Chart object.

PanelChart
public PanelChart(Imsl.Chart2D.Chart chart)

Description
Creates new PanelChart using a given Chart object.

Parameter
chart – A Chart to be displayed in this panel.

Methods

OnPaint

Description
Calls the UI delegate’s Paint method, if the UI delegate is non-null.

Parameter
painteventargs – The PaintEventArgs with the Graphics property for painting the chart.

Remarks
We pass the delegate a copy of the Graphics object to protect the rest of the Paint code from irrevocable changes (for example, Graphics.translate).

If you override this in a subclass you should not make permanent changes to the passed in Graphics.
For example, you should not alter the clip Rectangle or modify the transform. If you need to do these operations you may find it easier to create a new Graphics from the passed in Graphics and manipulate it.

Further, if you do not invoke super’s implementation you must honor the opaque property, that is if this component is opaque, you must completely fill in the background in a non-opaque color. If you do not honor the opaque property you will likely see visual artifacts.

OnResize
override void OnResize(System.EventArgs eventargs)

Description
When the PanelChart is resized, Refresh() is called.

Parameter
eventargs – The EventArgs.

Print
virtual public void Print()
Description
Print the Chart centered on a page.

---

**DrawPick Class**

public class Imsl.Chart2D.DrawPick : Draw

The DrawPick class.

**Properties**

**Node**
override public Imsl.Chart2D.ChartNode Node {set; }  

*Description*  
Specifies the current node of the chart tree.  

*Property Value*  
A ChartNode containing the current node of this tree.  

*Remarks*  
This is used to get drawing attributes from the tree.  

**Tolerance**
virtual public int Tolerance {get; set; }

*Description*  
The minimum distance that an event can be from a point or a line and still be considered a hit.  

*Property Value*  
An int containing the minimum distance that an event can be from a point or a line and still be considered a hit.

**Constructor**

**DrawPick**
public DrawPick(System.Windows.Forms MouseEventArgs mouseEventArgs,  
**Description**

Contracts a `DrawPick` object.

**Parameters**

- `mouseEventArgs` – A `MouseEvent` that provides data for the MouseUp, MouseDown, and MouseMove events.
- `graphics` – A `Graphics` object encapsulating a GDI+ drawing surface.
- `bounds` – A `Size` specifying the width and height of a rectangle.

**Methods**

**DrawArc**

```csharp
override public void DrawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

**Description**

Draws the outline of a circular or elliptical arc covering the specified rectangle.

**Parameters**

- `x` – An `int` which contains the x-coordinate of the upper-left corner of the rectangle that defines the ellipse.
- `y` – An `int` which contains the y-coordinate of the upper-left corner of the rectangle that defines the ellipse.
- `width` – An `int` which contains the width of the rectangle that defines the ellipse.
- `height` – An `int` which contains the height of the rectangle that defines the ellipse.
- `startAngle` – An `int` which specifies the angle in degrees measured clockwise from the x-axis to the starting point of the arc.
- `arcAngle` – An `int` which specifies an angle in degrees measured counter-clockwise from the `startAngle` parameter to the ending point of the arc.

**Remarks**

The center of the arc is center of this rectangle.

`startAngle = 0` is equivalent to the 3-o’clock position.

`DrawArc` draws the arc from `startAngle` to `startAngle + arcAngle`. A positive `arcAngle` indicates a counter-clockwise rotation. A negative `arcAngle` implies a clockwise rotation.

**DrawErrorBar**

```csharp
override public void DrawErrorBar(int x0, int y0, int x1, int y1, int flag)
```

**Description**

Draws an error bar.
Parameters
x0 – An int which specifies the x-coordinate of the beginning reference point.
y0 – An int which specifies the y-coordinate of the beginning reference point.
x1 – An int which specifies the x-coordinate of the ending reference point.
y1 – An int which specifies the y-coordinate of the ending reference point.
flag – An int that indicates which caps to draw.

DrawImage
override public void DrawImage(System.Drawing.Image image, int x, int y)

Description
Draws the specified image at the location specified by a coordinate pair.

Parameters
image – The Image object to draw.
x – An int which specifies the x-coordinate of the upper-left corner of the drawn image.
y – An int which specifies the x-coordinate of the upper-left corner of the drawn image.

DrawLine
override public void DrawLine(int x0, int y0, int x1, int y1)

Description
Draws a line from between two points.

Parameters
x0 – An int which specifies the x-coordinate of the line origin, (x0,y0).
y0 – An int which specifies the y-coordinate of the line origin, (x0,y0).
x1 – An int which specifies the x-coordinate of the line destination, (x1,y1).
y1 – An int which specifies the y-coordinate of the line destination, (x1,y1).

DrawMarker
override public void DrawMarker(int x, int y)

Description
Draws a marker.

Parameters
x – An int which specifies the x-coordinate of the marker destination, (x,y).
y – An int which specifies the y-coordinate of the marker destination, (x,y).

DrawText
override public System.Drawing.Size DrawText(Imsl.Chart2D.Text text, int x, int y)
Description
Draws a Text object.

Parameters
- text – A Text object to be drawn.
- x – An int which specifies the abscissa of the (x,y) point at which to start drawing the text.
- y – An int which specifies the ordinate of the (x,y) point at which to start drawing the text.

Returns
A Size containing the bounds of the Text to be drawn.

EndErrorBar
override public void EndErrorBar()

Description
Finish drawing an error bar.

EndFill
override public void EndFill()

Description
Finish drawing a filled region.

EndImage
override public void EndImage()

Description
Finish drawing an image.

EndLine
override public void EndLine()

Description
Finish drawing lines.

EndMarker
override public void EndMarker()

Description
Finish drawing markers.

EndText
override public void EndText()
**Description**
Finish drawing text.

**FillArc**
```java
override public void FillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

**Description**
Fills a circular or elliptical arc covering the specified rectangle.

**Parameters**
- **x** – An int which specifies the x-coordinate of the upper-left corner of the rectangular region that defines the ellipse from which the arc is drawn.
- **y** – An int which specifies the y-coordinate of the upper-left corner of the rectangular region that defines the ellipse from which the arc is drawn.
- **width** – An int which specifies the width of the rectangular region that defines the ellipse from which the arc is drawn.
- **height** – An int which specifies the height of the rectangular region that defines the ellipse from which the arc is drawn.
- **startAngle** – An int which specifies the starting angle of the arc, measured in degrees clockwise from the x-axis.
- **arcAngle** – An int which specifies an angle in degrees measured counter-clockwise from the `startAngle` parameter to the ending point of the arc.

**Remarks**
The center of the arc is center of this rectangle.

`startAngle = 0` is equivalent to the 3-o’clock position.

`DrawArc` draws the arc from `startAngle` to `startAngle + arcAngle`. A positive `arcAngle` indicates a counter-clockwise rotation. A negative `arcAngle` implies a clockwise rotation.

**FillPolygon**
```java
override public void FillPolygon(System.Drawing.Drawing2D.GraphicsPath polygon)
```

**Description**
Fill a polygon defined by a `Polygon` object.

**Parameter**
- **polygon** – A `Polygon` object which specifies the polygon to be filled.

**FillPolygon**
```java
override public void FillPolygon(int[] xpoints, int[] ypoints, int npoints)
```

**Description**
Fills a polygon.
Parameters

xpoints – An int array which contains the abscissae of the points which define the polygon.
ypoints – An int array which contains the ordinates of the points which define the polygon.
npoints – An int which specifies the number of points to add to the graphics path.

FillRectangle
override public void FillRectangle(int x, int y, int width, int height)

Description
Fill a rectangle.

Parameters

x – An int which specifies the x-coordinate of the upper-left corner of the rectangle.
y – An int which specifies the y-coordinate of the upper-left corner of the rectangle.
width – An int which specifies the width of the rectangle.
height – An int which specifies the height of the rectangle.

Fire
virtual public void Fire()

Description
Invoke the delegates for all of the picked nodes.

StartErrorBar
override public void StartErrorBar()

Description
Start drawing an error bar.

StartFill
override public void StartFill()

Description
Start drawing a filled region.

StartImage
override public void StartImage()

Description
Start drawing an image.

StartLine
override public void StartLine()
Start drawing lines.

StartMarker
override public void StartMarker()

Start drawing markers.

StartText
override public void StartText()

Start drawing text.

Translate
override public void Translate(int x, int y)

Prepends the specified translation to the transformation matrix of this Graphics object.

Parameters
- x – An int which specifies $dx$, the x component of the translation.
- y – An int which specifies $dy$, the y component of the translation.

PickEventArgs Class

public class Imsl.Chart2D.PickEventArgs : MouseEventArgs

An event that indicates that a chart element has been selected.
Provides data for the PickPerformed event.

See Also

(p. 1413)

Property

Node
virtual public Imsl.Chart2D.ChartNode Node {get; set; }
Description
The ChartNode associated with the pick event.

Property Value
A ChartNode which specifies the node associated with the pick event.

Constructor

PickEventArgs
public PickEventArgs(System.Windows.Forms MouseEventArgs mouseEvent)

Description
Initializes a new instance of the PickEventArgs class.

Parameter
- mouseEvent – A MouseEventArgs that provides data for the MouseUp, MouseDown, and MouseMove events.

Method

PointToLine
static public double PointToLine(int Px, int Py, int[] devA, int[] devB)

Description
Compute the distance from the point (Px, Py) to the line segment AB.

Parameters
- Px – An int which specifies the x coordinate of the point (Px, Py).
- Py – An int which specifies the y coordinate of the point (Px, Py).
- devA – An int[] which specifies the point that defines the head of the line segment.
- devB – An int[] which specifies the point that defines the tail of the line segment.

Returns
A double which contains the distance from the point (Px, Py) to the line segment AB.

Remarks
If the closest point from P to the line AB is not between A and B then the distance to the closer of A and B is returned.
PickEventHandler Delegate

public delegate Imsl.Chart2D.PickEventHandler
PickEventHandler

The delegate for receiving pick events. Represents the method that will handle the PickPerformed event of a chart node.

See Also
Imsl.Chart2D.PickEventArgs (p. 1411)

WebChart Class

public class Imsl.Chart2D.WebChart : Panel :
System.ComponentModel.IContainer, System.IDisposable,
System.Web.UI.IParserAccessor, System.Web.UI.IUrlResolutionService,
System.Web.UI.IDataBindingsAccessor, System.Web.UI.IContainerBuilderAccessor,
System.Web.UI.IContainerDesignerAccessor, System.Web.UI.IExpressionsAccessor,
System.Web.UI.IAttributeAccessor

A WebChart provides a component to use in ASP.NET applications that holds a Chart object.

Property

Chart

public Imsl.Chart2D.Chart Chart {get; set; }

Description
The Chart object associated with this WebChart.

Property Value
A Chart where the drawing is to rendered.
Constructor

**WebChart**

```csharp
public WebChart()
```

**Description**

Default constructor.

Methods

**OnInit**

```csharp
override void OnInit(System.EventArgs e)
```

**Description**

Initializes the object.

**Parameter**

- `e` – The EventArgs object that contains the event data.

**Render**

```csharp
override void Render(System.Web.UI.HtmlControls.HtmlTextWriter output)
```

**Description**

Renders the WebChart to the specified HTML writer.

**Parameter**

- `output` – The HtmlTextWriter that receives the control content.

DrawMap Class

```csharp
public class Imsl.Chart2D.DrawMap : Draw
```

Creates an HTML client-side imagemap from a chart tree.

Entries in the imagemap correspond to nodes that define the HREF attribute.

Properties

**Map**

```csharp
virtual public string Map { get; }
```

IMSL C# Numerical Library
Description
Returns the body of the HTML imagemap.

Property Value
A string that contains the body of the HTML client-side imagemap. The actual <map> and </map> tags are not included, so that the client code can more easily add attributes to the <map> tag.

Node
override public Imsl.Chart2D.ChartNode Node {set; }

Description
Specifies the current node of the chart tree.

Property Value
A ChartNode containing the current node of this tree.

Remarks
This is used to get drawing attributes from the tree.

Tolerance
virtual public int Tolerance {get; set; }

Description
The minimum distance that an event can be from a point or a line and still be considered a hit.

Property Value
An int containing the minimum distance that an event can be from a point or a line and still be considered a hit.

Constructor

DrawMap

Description
Contracts a DrawMap object.

Parameters
  graphics – A Graphics context in which to draw.
  bounds – A Size object containing the width and height of the chart to be drawn.
**Methods**

**DrawArc**

```java
override public void DrawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

**Description**

Draws the outline of a circular or elliptical arc covering the specified rectangle.

**Parameters**

- **x** – An int which contains the x-coordinate of the upper-left corner of the rectangle that defines the ellipse.
- **y** – An int which contains the y-coordinate of the upper-left corner of the rectangle that defines the ellipse.
- **width** – An int which contains the width of the rectangle that defines the ellipse.
- **height** – An int which contains the height of the rectangle that defines the ellipse.
- **startAngle** – An int which specifies the angle in degrees measured clockwise from the x-axis to the starting point of the arc.
- **arcAngle** – An int which specifies an angle in degrees measured counter-clockwise from the startAngle parameter to the ending point of the arc.

**Remarks**

The center of the arc is center of this rectangle. 
**startAngle** = 0 is equivalent to the 3-o’clock position.

**DrawArc** draws the arc from **startAngle** to **startAngle** + **arcAngle**. A positive **arcAngle** indicates a counter-clockwise rotation. A negative **arcAngle** implies a clockwise rotation.

**DrawErrorBar**

```java
override public void DrawErrorBar(int x0, int y0, int x1, int y1, int flag)
```

**Description**

Draws an error bar.

**Parameters**

- **x0** – An int which specifies the x-coordinate of the beginning reference point.
- **y0** – An int which specifies the y-coordinate of the beginning reference point.
- **x1** – An int which specifies the x-coordinate of the ending reference point.
- **y1** – An int which specifies the y-coordinate of the ending reference point.
- **flag** – An int that indicates which caps to draw.

**Remarks**

Legal values: 0=none, 1=bottom, 2=top, 3=both

**DrawImage**

```java
override public void DrawImage(System.Drawing.Image image, int x, int y)
```

**DrawMap**

IMSL C# Numerical Library
Description
Draws the specified image at the location specified by a coordinate pair.

Parameters
- image – The Image object to draw.
- x – An int which specifies the x-coordinate of the upper-left corner of the drawn image.
- y – An int which specifies the y-coordinate of the upper-left corner of the drawn image.

DrawLine
override public void DrawLine(int x0, int y0, int x1, int y1)

Description
Draws a line from between two points.

Parameters
- x0 – An int which specifies the x-coordinate of the line origin, (x0,y0).
- y0 – An int which specifies the y-coordinate of the line origin, (x0,y0).
- x1 – An int which specifies the x-coordinate of the line destination, (x1,y1).
- y1 – An int which specifies the y-coordinate of the line destination, (x1,y1).

DrawMarker
override public void DrawMarker(int x, int y)

Description
Draws a marker.

Parameters
- x – An int which specifies the x-coordinate of the marker destination, (x,y).
- y – An int which specifies the y-coordinate of the marker destination, (x,y).

EndErrorBar
override public void EndErrorBar()

Description
Finish drawing an error bar.

EndFill
override public void EndFill()

Description
Finish drawing a filled region.

EndImage
override public void EndImage()
**Description**
Finish drawing an image.

**EndLine**
override public void EndLine()

**Description**
Finish drawing lines.

**EndMarker**
override public void EndMarker()

**Description**
Finish drawing markers.

**EndText**
override public void EndText()

**Description**
Finish drawing text.

**FillArc**
override public void FillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

**Description**
Fills a circular or elliptical arc covering the specified rectangle.

**Parameters**

- **x** – An int which specifies the x-coordinate of the upper-left corner of the rectangular region that defines the ellipse from which the arc is drawn.
- **y** – An int which specifies the y-coordinate of the upper-left corner of the rectangular region that defines the ellipse from which the arc is drawn.
- **width** – An int which specifies the width of the rectangular region that defines the ellipse from which the arc is drawn.
- **height** – An int which specifies the height of the rectangular region that defines the ellipse from which the arc is drawn.
- **startAngle** – An int which specifies the starting angle of the arc, measured in degrees clockwise from the x-axis.
- **arcAngle** – An int which specifies an angle in degrees measured counter-clockwise from the **startAngle** parameter to the ending point of the arc.
Remarks
The center of the arc is center of this rectangle.
\( \text{startAngle} = 0 \) is equivalent to the 3-o’clock position.
\text{DrawArc} \text{ draws the arc from startAngle to startAngle+arcAngle.} \text{ A positive arcAngle indicates a counter-clockwise rotation. A negative arcAngle implies a clockwise rotation.}

\textbf{FillPolygon}
\texttt{override public void FillPolygon(System.Drawing.Drawing2D.GraphicsPath polygon)}

\textbf{Description}
Fill a polygon defined by a Polygon object.

\textbf{Parameter}
\begin{itemize}
  \item \texttt{polygon} – A Polygon object which specifies the polygon to be filled.
\end{itemize}

\textbf{FillPolygon}
\texttt{override public void FillPolygon(int[] xpoints, int[] ypoints, int npoints)}

\textbf{Description}
Fills a polygon.

\textbf{Parameters}
\begin{itemize}
  \item \texttt{xpoints} – An int array which contains the abscissae of the points which define the polygon.
  \item \texttt{ypoints} – An int array which contains the ordinates of the points which define the polygon.
  \item \texttt{npoints} – An int which specifies the number of points to add to the graphics path.
\end{itemize}

\textbf{FillRectangle}
\texttt{override public void FillRectangle(int x, int y, int width, int height)}

\textbf{Description}
Fill a rectangle.

\textbf{Parameters}
\begin{itemize}
  \item \texttt{x} – An int which specifies the x-coordinate of the upper-left corner of the rectangle.
  \item \texttt{y} – An int which specifies the y-coordinate of the upper-left corner of the rectangle.
  \item \texttt{width} – An int which specifies the width of the rectangle.
  \item \texttt{height} – An int which specifies the height of the rectangle.
\end{itemize}

\textbf{StartErrorBar}
\texttt{override public void StartErrorBar()}\texttt{()}

\textbf{Description}
Start drawing an error bar.

\textbf{StartFill}
\texttt{override public void StartFill()}\texttt{()}

\textbf{Chart2D}
**StartImage**

override public void StartImage()

**Description**

Start drawing an image.

**StartLine**

override public void StartLine()

**Description**

Start drawing lines.

**StartMarker**

override public void StartMarker()

**Description**

Start drawing markers.

**StartText**

override public void StartText()

**Description**

Start drawing text.

**Translate**

override public void Translate(int x, int y)

**Description**

Prepends the specified translation to the transformation matrix of this Graphics object.

**Parameters**

- x – An int which specifies dx, the x component of the translation.
- y – An int which specifies dy, the y component of the translation.

---

**BoxPlot Class**

```csharp
public class Imsl.Chart2D.BoxPlot : Data

Draws a multiple-group Box plot.
```

---

**IMSL C# Numerical Library**
For each group of observations, the box limits represent the lower quartile (25th percentile) and upper quartile (75th percentile). The median is displayed as a line across the box. Whiskers are drawn from the upper quartile to the upper adjacent value, and from the lower quartile to the lower adjacent value.

Optional notches may be displayed to show a 95 percent confidence interval about the median, at \( \pm 1.58 \frac{IRQ}{\sqrt{n}} \), where \( IRQ \) is the interquartile range and \( n \) is the number of observations. Outside and far outside values may be displayed as symbols. Outside values are outside the inner fence. Far out values are outside the outer fence.

The BoxPlot has several child nodes. Any of these nodes can be disabled by setting their “IsVisible” attribute to false.

- The “Bodies” attribute has the main body of the box plot elements. Its fill attributes determine the drawing of (notched) rectangle. Its line attributes determine the drawing of the median line. The width of the box is controlled by the “MarkerSize” attribute.
- The “Whiskers” attribute draws the lines to the upper and lower quartile. Its drawing is affected by the marker attributes.
- The “FarMarkers” attribute hold the far markers. Its drawing is affected by the marker attributes.
- The “OutsideMarkers” attribute hold the outside markers. Its drawing is affected by the marker attributes.

Fields

<table>
<thead>
<tr>
<th>BOXPLOTTYPEHORIZONTAL</th>
<th>public int BOXPLOTTYPEHORIZONTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Value for attribute “BoxPlotType” indicating that this is a horizontal box plot.</td>
</tr>
<tr>
<td>Remarks</td>
<td>Used in connection with BoxPlot nodes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BOXPLOTTYPEVERTICAL</th>
<th>public int BOXPLOTTYPEVERTICAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Value for attribute “BoxPlotType” indicating that this is a horizontal box plot.</td>
</tr>
<tr>
<td>Remarks</td>
<td>Used in connection with BoxPlot nodes.</td>
</tr>
</tbody>
</table>
Properties

**Bodies**

virtual public Imsl.Chart2D.ChartNode Bodies {get; }

**Description**
The main body of the BoxPlot elements.

**Property Value**
A ChartNode which contains the “Bodies” attribute value.

**BoxPlotType**

virtual public int BoxPlotType {get; set; }

**Description**
Specifies the orientation of the BoxPlot.

**Property Value**
An int which contains the “BoxPlotType” attribute value.

**Remarks**
Legal values are Imsl.Chart2D.BoxPlot.BOXPLOT_TYPE_VERTICAL (p. 1421) or Imsl.Chart2D.BoxPlot.BOXPLOT_TYPE_HORIZONTAL (p. 1421).

**FarMarkers**

virtual public Imsl.Chart2D.ChartNode FarMarkers {get; }

**Description**
The far markers of the BoxPlot elements.

**Property Value**
A ChartNode which contains the “FarMarkers” attribute value.

**Notch**

virtual public bool Notch {get; set; }

**Description**
Specifies whether the optional notches, indicating the extent of data falling within the 95 percent confidence range, are displayed.

**Property Value**
A bool which contains the “Notch” attribute value.

**Remarks**
true indicates that notches are to be displayed. default: false

**OutsideMarkers**

virtual public Imsl.Chart2D.ChartNode OutsideMarkers {get; }
Description
The outside markers of the BoxPlot elements.

Property Value
A ChartNode which contains the “OutsideMarkers” attribute value.

ProportionalWidth
virtual public bool ProportionalWidth {get; set; }

Description
Specifies whether the box widths are to be proportional.

Property Value
A bool which contains the “ProportionalWidth” attribute value.

Remarks
true indicates the box widths are to be proportional to the square root of the number of observations. If false all of the boxes have the same width. Default: false

Whiskers
virtual public Imsl.Chart2D.ChartNode Whiskers {get; }

Description
The wiskers of the BoxPlot elements drawn to the upper and lower quartile.

Property Value
A ChartNode which contains the “Whiskers” attribute value.

Constructors

BoxPlot
public BoxPlot(Imsl.Chart2D.AxisXY axis, double[] x, double[][] obs)

Description
Constructs a box plot chart node with specified x values.

Parameters
   axis – An AxisXY which is the parent of this node.
   x – A double[] which contains the x values.
   obs – A double[][] which contains the observations for each x.

Remarks
The number of rows in obs must equal the length of x. The length of each row in obs must be at least 4.
Description
Constructs a box plot chart node with specified x values.

Parameters
- axis – An AxisXY which is the parent of this node.
- x – a double[] which contains the x values.
- statistics – A BoxPlot.Statistics[] containing the statistics for each element in x.

Remarks
The number of BoxPlot.Statistics[] must equal x.Length.

BoxPlot
public BoxPlot(Imsl.Chart2D.AxisXY axis, double[][] obs)

Description
Constructs a box plot chart.

Parameters
- axis – An AxisXY which is the parent of this node.
- obs – A double[] containing the observations.

Remarks
The length of each row in obs must be at least 4.

Methods

GetStatistics
virtual public Imsl.Chart2D.BoxPlot.Statistics[] GetStatistics() (BoxPlot)

Description
Returns statistics for each set of observations.

Returns
A BoxPlot.Statistics[] containing the statistics for each set of observations.

GetStatistics
virtual public Imsl.Chart2D.BoxPlot.Statistics GetStatistics(int iSet) (BoxPlot)

Description
Returns statistics for a set of observations.

Parameter
- iSet – An int which specifies the index of a set whose statistics are to be returned.
Returns
A BoxPlot.Statistics containing the statistics for the iSet set of observations.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetDataRange
override public void SetDataRange(double[] range)

Description
Update the data range, range = \{xmin, xmax, ymin, ymax\}

Parameter
range – a double array which contains the updated range, \{xmin, xmax, ymin, ymax\}

Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

SetLabels
virtual public void SetLabels(string[] labels)

Description
Sets up an axis with labels.

Parameter
labels – A String[] containing the axis labels.

Remarks
Sets up an axis with labels. This turns off the tick marks and sets the “BoxPlotType” attribute. It also turns off autoscaling for the axis and sets its “Window” and “Number” and “Ticks” attribute as appropriate for a labeled Box plot. The existing value of the “BoxPlotType” attribute is used to determine the axis to be modified.

SetLabels
virtual public void SetLabels(string[] labels, int type)
Sets up an axis with labels.

**Parameters**

- `labels` – A String[] containing the axis labels.
- `type` – An int which specifies the “BoxPlotType” attribute value.

**Remarks**

This turns off the tick marks and sets the “BoxPlotType” attribute. It also turns off autoscaling for the axis and sets its “Window”, “Number” and “Ticks” attributes as appropriate for a labeled Box plot. The number of labels must equal the number of items.

Legal values for `type` are `Imsl.Chart2D.BoxPlot.BOXPLOT_TYPE_VERTICAL` (p. 1421) or `Imsl.Chart2D.BoxPlot.BOXPLOT_TYPE_HORIZONTAL` (p. 1421). This determines the axis to be modified.

**Example: Box Plot Chart**

A simple box plot chart is constructed in this example. Display of far and outside values is turned on.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class BoxPlotEx1 : FrameChart
{
    public BoxPlotEx1()
    {
        Chart chart = this.Chart;

        double[][] obs = {new double[]{66.0, 52.0, 49.0, 64.0, 68.0, 26.0, 86.0,
                                52.0, 43.0, 75.0, 87.0, 188.0, 118.0,
                                103.0, 82.0, 71.0, 103.0, 240.0, 31.0,
                                40.0, 47.0, 51.0, 31.0, 47.0, 14.0,
                                71.0},
                       new double[]{61.0, 47.0, 196.0, 131.0, 173.0, 37.0, 47.0,
                                215.0, 230.0, 69.0, 98.0, 125.0, 94.0,
                                72.0, 72.0, 125.0, 143.0, 192.0, 122.0,
                                32.0, 114.0, 32.0, 23.0, 71.0, 38.0,
                                136.0, 169.0},
                       new double[]{152.0, 201.0, 134.0, 206.0, 92.0, 101.0,
                                     119.0, 124.0, 133.0, 83.0, 60.0, 124.0,
                                     142.0, 124.0, 64.0, 75.0, 103.0, 46.0,
                                     68.0, 87.0, 27.0, 73.0, 59.0, 119.0, 64.0,
                                     111.0},
                       new double[]{80.0, 68.0, 24.0, 24.0, 82.0, 100.0, 55.0,
                                     91.0, 87.0, 64.0, 170.0, 86.0, 202.0,
                                     71.0, 85.0, 122.0, 155.0, 80.0, 71.0,
                                     28.0, 212.0, 80.0, 24.0, 80.0, 169.0},
                       new double[]{66.0, 52.0, 49.0, 64.0, 68.0, 26.0, 86.0,
                                     52.0, 43.0, 75.0, 87.0, 188.0, 118.0,
                                     103.0, 82.0, 71.0, 103.0, 240.0, 31.0,
                                     40.0, 47.0, 51.0, 31.0, 47.0, 14.0,
                                     71.0},
                       new double[]{61.0, 47.0, 196.0, 131.0, 173.0, 37.0, 47.0,
                                     215.0, 230.0, 69.0, 98.0, 125.0, 94.0,
                                     72.0, 72.0, 125.0, 143.0, 192.0, 122.0,
                                     32.0, 114.0, 32.0, 23.0, 71.0, 38.0,
                                     136.0, 169.0}};
```
double[] x = new double[]{1.0, 2.0, 3.0, 4.0, 5.0};

// Create an instance of a BoxPlot Chart
AxisXY axis = new AxisXY(chart);
BoxPlot boxPlot = new BoxPlot(axis, obs);
boxPlot.SetLabels(xLabels);

// Customize the fill color and the outside and far markers
boxPlot.Bodies.FillColor = System.Drawing.Color.FromName("blue");
boxPlot.OutsideMarkers.MarkerType = Imsl.Chart2D.BoxPlot.MARKER_TYPE_HOLLOW_CIRCLE;
boxPlot.OutsideMarkers.MarkerColor = System.Drawing.Color.FromName("purple");
boxPlot.FarMarkers.MarkerType = Imsl.Chart2D.BoxPlot.MARKER_TYPE_ASTERISK;
boxPlot.FarMarkers.MarkerColor = System.Drawing.Color.FromName("red");

// Set titles
chart.ChartTitle.SetTitle("Ozone Levels in Stanford by Month");
axis.AxisX.AxisTitle.SetTitle("Month");
axis.AxisY.AxisTitle.SetTitle("Ozone Level");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new BoxPlotEx1());
}
}
BoxPlot.Statistics Class

public class Imsl.Chart2D.BoxPlot.Statistics

Computes the statistics for one set of observations in a Boxplot.
Properties

**LowerAdjacentValue**

```csharp
virtual public double LowerAdjacentValue {get; }
```

**Description**

A double which contains the lower adjacent value.

**LowerQuartile**

```csharp
virtual public double LowerQuartile {get; }
```

**Description**

A double which contains the lower quartile value (25th percentile).

**MaximumValue**

```csharp
virtual public double MaximumValue {get; }
```

**Description**

A double which contains the maximum value of this set.

**Median**

```csharp
virtual public double Median {get; }
```

**Description**

A double which contains the median value for a set of observations.

**MedianLowerConfidenceInterval**

```csharp
virtual public double MedianLowerConfidenceInterval {get; }
```

**Description**

A double which contains the lower confidence interval for the median value of this set of observations.

**MedianUpperConfidenceInterval**

```csharp
virtual public double MedianUpperConfidenceInterval {get; }
```

**Description**

A double which contains the upper confidence interval for the median value of this set of observations.

**MinimumValue**

```csharp
virtual public double MinimumValue {get; }
```

**Description**

A double which contains the minimum value of this set.

**NumberObservations**

```csharp
virtual public int NumberObservations {get; }
```
Description
An int which contains the number of observations in this set.

**UpperAdjacentValue**
virtual public double UpperAdjacentValue {get; }

Description
A double which contains the upper adjacent value.

**UpperQuartile**
virtual public double UpperQuartile {get; }

Description
A double which contains the upper quartile value (75th percentile).

**Constructor**

**Statistics**
public Statistics(double[] obs)

Description
Creates a new instance of BoxPlot.Statistics.

Parameter
obs – A double[] containing the set of observations.

Remarks
There must be at least 4 observations to compute the statistics.

Exception
System.ArgumentException is thrown if there are fewer than 4 observations.

**Methods**

**GetFarMarkers**
virtual public double[] GetFarMarkers()

Description
Returns the far markers.
Returns
A double[] which contains the far markers for this set.

GetOutsideMarkers
virtual public double[] GetOutsideMarkers()

Description
Returns the outside markers.

Returns
A double[] which contains the outside markers for this set.

Contour Class

public class Imsl.Chart2D.Contour : Data
A Contour chart shows level curves of a two-dimensional function.

The function can be defined either as values on a rectangular grid or by scattered data points.

A set of ContourLevel (p. 1445) objects are created as children of this node. The number of ContourLevels is one more than the number of level curves. If the level curve values are \( c_0, \ldots, c_{n-1} \) then the \( k\)-th ContourLevel child corresponds to \( c_{k-1} < z \leq c_k \).

To change the look of the contour chart, change the line attributes (specified with LineColor (p. 1302), LineWidth (p. 1303) and SetMarkerDashPattern (p. 1331)) and fill attributes (specified with FillType (p. 1322) and FillColor (p. 1301)) in the ContourLevel nodes.

A Legend object is also created as a child of this node. It should be used instead of the usual chart legend. By default, this legend is not shown. To show it, set isVisible = true.

See Also
Imsl.Chart2D.ContourLevel (p. 1445)

Property

ContourLegend
virtual public Imsl.Chart2D.Contour.Legend ContourLegend {get; }
Description
Contains the legend information associated with this Contour.

Property Value
A Legend object for this Contour.

Remarks
By default, the legend is not drawn because IsVisible is set to false. To show the legend set IsVisible = true, i.e., contour.ContourLegend.IsVisible = true;

 Constructors

Contour
public Contour(Imsl.Chart2D.AxisXY axis, double[] xGrid, double[] yGrid, double[,] zData, double[] cLevel)

Description
Creates a Contour chart from rectangularly gridded data.

Parameters
axis – An AxisXY containing the parent node of this Contour.
xGrid – A double[] which contains the x-coordinate values of the grid.
yGrid – A double[] which contains the y-coordinate values of the grid.
zData – A double[,] which contains the function values to be contoured.
cLevel – A double[] which contains the values of the contour levels.

Remarks
The value of the function at (xGrid[i],yGrid[j]) is given by zData[i][j]. The size of zData must be xGrid.Length by yGrid.Length.

Contour
public Contour(Imsl.Chart2D.AxisXY axis, double[] xGrid, double[] yGrid, double[,] zData)

Description
Creates a Contour chart from rectangularly gridded data with computed contour levels.

Parameters
axis – An AxisXY containing the parent node of this Contour.
xGrid – A double[] which contains the x-coordinate values of the grid.
yGrid – A double[] which contains the y-coordinate values of the grid.
zData – A double[,] which contains the function values to be contoured.
Remarks
The contour levels are chosen to span the data and to be “nice” values. The value of the function at (xGrid[i], yGrid[j]) is given by zData[i][j]. The size of zData must be xGrid.Length by yGrid.Length.

Contour
public Contour(Imsl.Chart2D.AxisXY axis, double[] x, double[] y, double[] z)

Description
Creates a Contour chart from scattered data with computed contour levels.

Parameters
axis – An AxisXY containing the parent node of this Contour.
x – A double[] which contains the x-values of the data points.
y – A double[] which contains the y-values of the data points.
z – A double[] which contains the x-values of the data points.

Remarks
The contour chart is created by using a radial basis approximation to estimate the functions value on a rectangular grid. The contour chart is then computed as for gridded data.

See Also: Imsl.Math.RadialBasis (p. 195)

Contour
public Contour(Imsl.Chart2D.AxisXY axis, double[] x, double[] y, double[] z, double[] cLevel, int nCenters)

Description
Creates a Contour chart from scattered data with computed contour levels.

Parameters
axis – An AxisXY containing the parent node of this Contour.
x – A double[] which contains the x-values of the data points.
y – A double[] which contains the y-values of the data points.
z – A double[] which contains the x-values of the data points.
cLevel – A double[] which contains the values of the contour levels.
nCenters – An int specifying the number of centers to use for the radial basis approximation.

Remarks
The contour chart is created by using a radial basis approximation to estimate the functions value on a rectangular grid. The contour chart is then computed as for gridded data.
A larger number of centers will provide a closer, but noiser approximation.

See Also: Imsl.Math.RadialBasis (p. 195)
Methods

GetContourLevel
virtual public Imsl.Chart2D.ContourLevel[] GetContourLevel()

Description
Returns all of the contour levels.

Returns
A ContourLevel[] containing the “ContourLevel” attribute value.

GetContourLevel
virtual public Imsl.Chart2D.ContourLevel GetContourLevel(int k)

Description
Returns a specified ContourLevel.

Parameter
k – An int which indicates what ContourLevel to return.

Returns
A ContourLevel that corrisponds to the k-th level (cLevel[k]).

Remarks
The k-th contour level contains the level curve equal to cLevel[k] in the constructor. It also contains the fill areas for the values in the interval (cLevel[k-1], cLevel[k]).
The first contour level (k=0) contains the fill area for values less than cLevel[0] and the level curves lines where the function value equals cLevel[0].
The last contour level (k=cLevel.Length) contains the fill area for values greater than cLevel[cLevel.length-1], but no level curve lines.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetDataRange
override public void SetDataRange(double[] range)
Description
Update the data range, range = \{xmin,xmax,ymin,ymax\}

Parameter
range – a double array which contains the updated range, \{xmin,xmax,ymin,ymax\}

Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

Example: Contour Chart from Gridded Data

In the restricted three-body problem, two large objects (masses \(M_1\) and \(M_2\)) a distance \(a\) apart, undergoing mutual gravitational attraction, circle a common center-of-mass. A third small object (mass \(m\)) is assumed to move in the same plane as \(M_1\) and \(M_2\) and is assumed to be too small to affect the large bodies. For simplicity, we use a coordinate system that has the center of mass at the origin. \(M_1\) and \(M_2\) are on the x-axis at \(x_1\) and \(x_2\), respectively.

In the center-of-mass coordinate system, the effective potential energy of the system is given by

\[
V = \frac{m(M_1 + M_2)}{a} \left[ \frac{x_2}{\sqrt{(x - x_1)^2 + y^2}} - \frac{x_1}{\sqrt{(x - x_2)^2 + y^2}} - \frac{1}{2} \left( x^2 + y^2 \right) \right]
\]

The universal gravitational constant is \(G\). The following program plots the part of \(V(x,y)\) inside of the square bracket. The factor \(\frac{m(M_1 + M_2)}{a}\) is ignored because it just scales the plot.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

class ContourEx1 : FrameChart
{
    public ContourEx1()
    {
        Chart chart = this.Chart;
        int nx = 80;
        int ny = 80;

        // Allocate space
        double[] xGrid = new double[nx];
        double[] yGrid = new double[ny];
        double[,] zData = new double[nx,ny];

        // Setup the grids points
        for (int i = 0; i < nx; i++)
        {
            xGrid[i] = -2 + 4.0 * i / (double) (nx - 1);
        }
        for (int j = 0; j < ny; j++)
        {
            yGrid[j] = -2 + 4.0 * j / (double) (ny - 1);
        }
    }
}
```
{ 
    yGrid[j] = - 2 + 4.0 * j / (double) (ny - 1);
}

// Evaluate the function at the grid points
for (int i = 0; i < nx; i++)
{
    for (int j = 0; j < ny; j++)
    {
        double x = xGrid[i];
        double y = yGrid[j];
        double rm = 0.5;
        double x1 = rm / (1.0 + rm);
        double x2 = x1 - 1.0;
        double d1 = Math.Sqrt((x - x1) * (x - x1) + y * y);
        double d2 = Math.Sqrt((x - x2) * (x - x2) + y * y);
        zData[i,j] = x2 / d1 - x1 / d2 - 0.5 * (x * x + y * y);
    }
}

// Create the contour chart, with user-specified levels and a legend
AxisXY axis = new AxisXY(chart);
double[] cLevel = new double[]{ -7, -5.4, -3, -2.3, -2.1, -1.97, -1.85,
                                  -1.74, -1.51, -1.39, -1};
Contour c = new Contour(axis, xGrid, yGrid, zData, cLevel);
c.ContourLegend.IsVisible = true;

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new ContourEx1());
}
Example: Contour Chart from Scattered Data

In this example, a contour chart is created from 150, randomly chosen, scattered data points. The function is $\sqrt{x^2 + y^2}$, so the level curve should be circles.

The input data is shown on top of the contours as small green circles. The chart data nodes are drawn in the order in which they are added, so the input data marker node has to be added to the axis after the contour, so that the markers are not hidden.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

class ContourEx2 : FrameChart
{
    public ContourEx2()
```
Chart chart = this.Chart;

int n = 150;

// Allocate space
double[] x = new double[n];
double[] y = new double[n];
double[] z = new double[n];

System.Random random = new System.Random((System.Int32)123457);

double[] randomValue = new double[150];
randomValue[0] = 0.41312962995625035;
randomValue[1] = 0.8225528716547005;
randomValue[2] = 0.44364905186692527;
randomValue[3] = 0.9887088342522812;
randomValue[4] = 0.9647868112234352;
randomValue[5] = 0.5668831243079411;
randomValue[6] = 0.27386697614898103;
randomValue[7] = 0.8805853693809824;
randomValue[8] = 0.718029622748057;
randomValue[9] = 0.6153607537410654;
randomValue[10] = 0.3158193853638753;
randomValue[11] = 0.10778643304578747;
randomValue[12] = 0.09275375134615693;
randomValue[13] = 0.9817642781628322;
randomValue[14] = 0.4673631886309925;
randomValue[15] = 0.9066980293517674;
randomValue[16] = 0.31440695305815347;
randomValue[17] = 0.99915607629566562;
randomValue[18] = 0.785150345014761;
randomValue[19] = 0.7930129038729785;
randomValue[20] = 0.5695413465811706;
randomValue[21] = 0.7625752595574732;
randomValue[22] = 0.482465474704169;
randomValue[23] = 0.09904819350827354;
randomValue[24] = 0.7013979421419555;
randomValue[25] = 0.8127581377189425;
randomValue[26] = 0.2160980302718407;
randomValue[27] = 0.2618716012466812;
randomValue[28] = 0.966175212476057;
randomValue[29] = 0.8929180151759015;
randomValue[30] = 0.9253777827882632;
randomValue[31] = 0.3192464623158826;
randomValue[32] = 0.6191390558809441;
randomValue[33] = 0.860615090126798;
randomValue[34] = 0.4202423262221493;
randomValue[35] = 0.3204336652731257;
randomValue[36] = 0.3501592792324697;
randomValue[37] = 0.08674811183862785;
randomValue[38] = 0.5605305915601296;
randomValue[39] = 0.6088802062708134;
randomValue[40] = 0.8382035138841133;
randomValue[41] = 0.923698754556213;
randomValue[42] = 0.8024356174828979;
double[] randomValueY=new double[150];
randomValueY[0]=.15995876895053263;
randomValueY[1]=.48794367683379836;
randomValueY[2]=.20896329070872555;
randomValueY[3]=0.4781765623804778;
randomValueY[4]=0.6732389937186418;
randomValueY[5]=0.33081942994459734;
randomValueY[6]=0.10880787186704965;
randomValueY[7]=0.901139442534768;
randomValueY[8]=0.48723656383264413;
randomValueY[9]=0.10153552805288812;
randomValueY[10]=0.9558058275075961;
randomValueY[11]=0.011829287599608884;
randomValueY[12]=0.4859902873228249;
randomValueY[13]=0.5505301300240635;
randomValueY[14]=0.1865244274911184;
randomValueY[15]=0.9272326533193322;
randomValueY[16]=0.4215880116306273;
randomValueY[17]=0.0386317648903991;
randomValueY[18]=0.6451521871931544;
randomValueY[19]=0.81930105474355;
randomValueY[20]=0.039285689951912395;
randomValueY[21]=0.31325564481720314;
randomValueY[22]=0.6272275622766595;
randomValueY[23]=0.8934533907186641;
randomValueY[24]=0.5212913217641422;
randomValueY[25]=0.6237725863035143;
randomValueY[26]=0.3611731793838059;
randomValueY[27]=0.23163547542978535;
randomValueY[28]=0.7999943624102621;
randomValueY[29]=0.5393342599409070;
randomValueY[30]=0.10341603798162413;
randomValueY[31]=0.48822476962455685;
randomValueY[32]=0.5414223626279245;
randomValueY[33]=0.08241640235000847;
randomValueY[34]=0.27287579835296155;
randomValueY[35]=0.6770805504344167;
randomValueY[36]=0.849705978782107;
randomValueY[37]=0.04142051621648373;
randomValueY[38]=0.30060172837976995;
randomValueY[39]=0.5378809821731352;
randomValueY[40]=0.993333184285308;
randomValueY[41]=0.5755163489718148;
randomValueY[42]=0.12033991348116369;
randomValueY[43]=0.22044795260992822;
randomValueY[44]=0.7039752563092764;
randomValueY[45]=0.47510550779825345;
randomValueY[46]=0.47581191139276346;
randomValueY[47]=0.2746412789430772;
randomValueY[48]=0.8486627562667742;
randomValueY[49]=0.6911278265254134;
randomValueY[50]=0.47048601468635676;
randomValueY[51]=0.1848034365963364;
randomValueY[52]=0.5260974820985063;
randomValueY[53]=0.9965118715946334;
randomValueY[54]=0.03562254706322543;
randomValueY[55]=0.9366159496862719;
randomValueY[56]=0.8878769321024975;
randomValueY[57]=0.8930475165444577;
randomValueY[58]=0.24237426250726957;
for (int k = 0; k < n; k++)
{
    x[k] = randomValue[k];
    y[k] = randomValueY[k];
    z[k] = System.Math.Sqrt(x[k] * x[k] + y[k] * y[k]);
}

// Setup the contour plot and its legend
AxisXY axis = new AxisXY(chart);
Contour contour = new Contour(axis, x, y, z);
contour.ContourLegend.IsVisible = true;

// Show the input data points as small green circles
Data dataPoints = new Data(axis, x, y);
dataPoints.DataType = Data.DATA_TYPE_MARKER;
dataPoints.MarkerType = Data.MARKER_TYPE_FILLED_CIRCLE;
dataPoints.MarkerColor = System.Drawing.Color.FromArgb(0, 255, 0);
dataPoints.MarkerSize = 0.5;
public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new ContourEx2());
}

Output

Contour.Legend Class

public class Imsl.Chart2D.Contour.Legend : AxisXY

1444 • Contour.Legend  IMSL C# Numerical Library
A legend for a contour chart.

This legend should be used for contour charts, instead of usual chart legend. The “Viewport” attribute for this node is set to [0.83,0.98] by [0.1,0.6].

**Method**

**Paint**

```java
override public void Paint(Imsl.Chart2D.Draw draw)
```

**Description**

Paints this node and all of its children.

**Parameter**

`draw` – A `Draw` which is to be painted.

**Remarks**

This is normally called only by the `Paint` method in this node’s parent.

---

**ContourLevel Class**

```java
public class Imsl.Chart2D.ContourLevel : ChartNode
```

ContourLevel draws a level curve line and the fill area between the level curve and the next smaller level curve.

ContourLevel objects are created by Contour as child nodes.

Each ContourLevel defines a filled areas and level curves. The drawing of the filled areas can be changed using the line attributes (specified with LineColor (p. 1302), LineWidth (p. 1303) and SetMarkerDashPattern (p. 1331)) and fill attributes (specified with FillType (p. 1322) and FillColor (p. 1301)) in the ContourLevel nodes.

**See Also**

Imsl.Chart2D.Contour (p. 1431)
**Method**

**Paint**
```java
override public void Paint(Imsl.Chart2D.Draw draw)
```

**Description**
Paints this node and all of its children.

**Parameter**
- `draw` – A Draw which is to be painted.

**Remarks**
This is normally called only by the `Paint` method in this node's parent.

---

**ErrorBar Class**

```java
public class Imsl.Chart2D.ErrorBar : Data
```

Renders data points with error bars.

**Fields**

**DATA_TYPE_ERROR_X**
```java
public int DATA_TYPE_ERROR_X
```

**Description**
Value for attribute “DataType” indicating that this is a horizontal error bar.

**Remarks**
Used in connection with ErrorBar nodes.

**DATA_TYPE_ERROR_Y**
```java
public int DATA_TYPE_ERROR_Y
```

**Description**
Value for attribute “DataType” indicating that this is a vertical error bar.

**Remarks**
Used in connection with ErrorBar nodes.
Constructor

ErrorBar
public ErrorBar(Imsl.Chart2D.AxisXY axis, double[] x, double[] y, double[] low, double[] high)

Description
Creates a set of error bars centered at \((x[k], y[k])\) and with extents \(\text{low}[k], \text{high}[k]\).

Parameters

axis – An Axis containing the parent of this node.

\(x\) – A double[] which contains the x-coordinates of the points at which the error bars will be centered.

\(y\) – A double[] which contains the y-coordinates of the points at which the error bars will be centered.

\(\text{low}\) – A double[] which contains the values which define the minimum extent of the error bars.

\(\text{high}\) – A double[] which contains the values which define the maximum extent of the error bars.

Remarks
If DataType (p. 1320) has the bit Imsl.Chart2D.ErrorBar.DATA_TYPE_ERROR_X (p. 1446) set then this is a horizontal error bar. If the bit Imsl.Chart2D.ErrorBar.DATA_TYPE_ERROR_Y (p. 1446) is set then this is a vertical error bar. If neither bit is set then no error bar is drawn.

A Data node with the same x and y values can be used to put markers at the center of each error bar.

Each of the array arguments have an associated attribute. That is, “X”, “Y”, “Low” and “High”.

Methods

GetHigh
virtual public double[] GetHigh()

Description
Returns the maximum extent of the error bars.

Returns
A double[] which contains the values for the maximum extent of the error bars.

GetLow
virtual public double[] GetLow()

Description
Returns the minimum extent of the error bars.
Returns
A double[] which contains the values for the minimum extent of the error bars.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetDataRange
override public void SetDataRange(double[] range)

Description
Update the data range, range = {xmin, xmax, ymin, ymax}

Parameter
range – a double array which contains the updated range, {xmin, xmax, ymin, ymax}

Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

SetHigh
virtual public void SetHigh(double[] high)

Description
Sets the maximum extent of the error bars.

Parameter
high – A double[] which contains the values for the maximum extent of the error bars.

SetLow
virtual public void SetLow(double[] low)

Description
Sets the minimum extent of the error bars.

Parameter
low – A double[] which contains the values for the minimum extent of the error bars.
Example: ErrorBar Chart

An ErrorBar chart is constructed in this example. Three data sets are used and a legend is added to the chart.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

generic class ErrorBarEx1 : FrameChart
{
    public ErrorBarEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
        double dx = .5 * Math.PI/(npoints - 1);
        double[] x = new double[npoints];
        double[] y1 = new double[npoints];
        double[] y2 = new double[npoints];
        double[] y3 = new double[npoints];
        double[] low1 = new double[npoints];
        double[] low2 = new double[npoints];
        double[] low3 = new double[npoints];
        double[] hi1 = new double[npoints];
        double[] hi2 = new double[npoints];
        double[] hi3 = new double[npoints];

        // Generate some data
        for (int i = 0; i < npoints; i++)
        {
            x[i] = i * dx;
            y1[i] = System.Math.Sin(x[i]);
            low1[i] = x[i] - .05;
            hi1[i] = x[i] + .05;
            y2[i] = System.Math.Cos(x[i]);
            low2[i] = y2[i] - .07;
            hi2[i] = y2[i] + .03;
            y3[i] = System.Math.Atan(x[i]);
            low3[i] = y3[i] - .01;
            hi3[i] = y3[i] + .04;
        }

        // Data
        Data d1 = new Data(axis, x, y1);
        Data d2 = new Data(axis, x, y2);
        Data d3 = new Data(axis, x, y3);

        // Set Data Type to Marker
        d1.DataType = Data.DATA_TYPE_MARKER;
        d2.DataType = Data.DATA_TYPE_MARKER;
        d3.DataType = Data.DATA_TYPE_MARKER;

        // Set Marker Types
    }
}
```
d1.MarkerType = Data.MARKER_TYPE_CIRCLE_PLUS;
d2.MarkerType = Data.MARKER_TYPE_HOLLOW_SQUARE;
d3.MarkerType = Data.MARKER_TYPE_ASTERISK;

// Set Marker Colors

// Create an instances of ErrorBars
ErrorBar ebar1 = new ErrorBar(axis, x, y1, low1, hi1);
ErrorBar ebar2 = new ErrorBar(axis, x, y2, low2, hi2);
ErrorBar ebar3 = new ErrorBar(axis, x, y3, low3, hi3);

// Set Data Type to Error_X
ebar1.DataType = ErrorBar.DATA_TYPE_ERROR_X;
ebar2.DataType = ErrorBar.DATA_TYPE_ERROR_Y;
ebar3.DataType = ErrorBar.DATA_TYPE_ERROR_Y;

// Set Marker Colors

// Set Data Labels
d1.SetTitle("Sine");
d2.SetTitle("Cosine");
d3.SetTitle("ArcTangent");

// Add a Legend
Legend legend = chart.Legend;
legend.SetTitle(new Text("Legend"));
legend.IsVisible = true;

// Set the Chart Title
chart.ChartTitle.SetTitle("Error Bar Plot");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new ErrorBarEx1());
}

IMSL C# Numerical Library
HighLowClose Class

public class Imsl.Chart2D.HighLowClose : Data
High-low-close plot of stock data.
Field

DAY
public double DAY

Description
Ticks per day.

Constructors

HighLowClose
public HighLowClose(Imsl.Chart2D.AxisXY axis, System.DateTime start, double[] high, double[] low, double[] close)

Description
Constructs a high-low-close chart node beginning with specified start date.

Parameters
axis – An Axis specifying the parent of this node.
start – A DateTime which specifies the first date.
high – A double[] which contains the stock’s high prices.
low – A double[] which contains the stock’s low prices.
close – A double[] which contains the stock’s closing prices.

Remarks
The high, low and close are used to specify the respective attributes. That is, “high”, “low” and “close”.

HighLowClose
public HighLowClose(Imsl.Chart2D.AxisXY axis, System.DateTime start, double[] high, double[] low, double[] close, double[] open)

Description
Constructs a high-low-close-open chart node beginning with specified start date.

Parameters
axis – An Axis specifying the parent of this node.
start – A DateTime which specifies the first date.
high – A double[] which contains the stock’s high prices.
low – A double[] which contains the stock’s low prices.
close – A double[] which contains the stock’s closing prices.
open – A double[] which contains the stock’s opening prices.
Remarks
The high, low, close and open are used to specify the respective attributes. That is, “high”, “low”, “close” and “open”.

HighLowClose
public HighLowClose(Imsl.Chart2D.AxisXY axis, double[] x, double[] high, double[] low, double[] close)

Description
Constructs a high-low-close chart node beginning with specified start date.

Parameters
   axis – An Axis specifying the parent of this node.
   x – A double[] which contains the axis points.
   high – A double[] which contains the stock’s high prices.
   low – A double[] which contains the stock’s low prices.
   close – A double[] which contains the stock’s closing prices.

Remarks
The X, high, low and close are used to specify the respective attributes. That is, “X”, “high”, “low” and “close”.

HighLowClose
public HighLowClose(Imsl.Chart2D.AxisXY axis, double[] x, double[] high, double[] low, double[] close, double[] open)

Description
Constructs a high-low-close-open chart node beginning with specified start date.

Parameters
   axis – An Axis specifying the parent of this node.
   x – A double array which contains the axis points.
   high – A double[] which contains the stock’s high prices.
   low – A double[] which contains the stock’s low prices.
   close – A double[] which contains the stock’s closing prices.
   open – A double[] which contains the stock’s opening prices.

Remarks
The X, high, low and close and open are used to specify the respective attributes. That is, “X”, “high”, “low”, “close” and “open”.

Chart2D	 HighLowClose • 1453
```csharp
Methods

GetClose
virtual public double[] GetClose()

Description
Returns the stock prices at close.

Returns
A double[] containing the closing stock prices.

GetHigh
virtual public double[] GetHigh()

Description
Returns the high stock prices.

Returns
A double[] containing the high stock prices.

GetLow
virtual public double[] GetLow()

Description
Returns the low stock prices.

Returns
A double[] containing the low stock prices.

GetOpen
virtual public double[] GetOpen()

Description
Returns the opening stock prices.

Returns
A double[] containing the opening stock prices.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.
```
Remarks
This is normally called only by the Paint method in this node’s parent.

SetClose
virtual public void SetClose(double[] close)

Description
Sets the closing stock prices.

Parameter
  close – A double[] specifying the closing stock prices.

SetDataRange
override public void SetDataRange(double[] range)

Description
Update the data range, range = {xmin, xmax, ymin, ymax}

Parameter
  range – a double array which contains the updated range, {xmin, xmax, ymin, ymax}

Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

SetDateAxis
virtual public void SetDateAxis(string labelFormat)

Description
Sets up the x-axis for high-low-close plot.

Parameter
  labelFormat – A string used to format the date axis labels.

Remarks
This turns off autoscaling on the x-axis and sets the “Window” attribute depending on the number of dates being plotted. The Number attribute determines the number of intervals along the x-axis.

The labelFormat sets TextFormat (p. 1305) and TextFormatProvider (p. 1305) in the Imsl.Chart2D.AxisLabel (p. 1358) node.

SetHigh
virtual public void SetHigh(double[] high)

Description
Sets the high stock prices.
Parameter
   high – A double[] specifying the high stock prices.

SetLow
virtual public void SetLow(double[] low)

Description
Sets the low stock prices.

Parameter
   low – A double[] specifying the low stock prices.

SetOpen
virtual public void SetOpen(double[] open)

Description
Sets the opening stock prices.

Parameter
   open – A double[] specifying the opening stock prices.

Example: High-Low-Close Chart
A simple high-low-close chart is constructed in this example.
Autoscaling does not properly handle time data, so autoscaling is turned off for the x (time) axis and the axis limits are set explicitly.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class HiLoEx1 : FrameChart
{
    public HiLoEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);
        // Date is June 27, 1999
        System.Globalization.GregorianCalendar temp_calendar;
        temp_calendar = new System.Globalization.GregorianCalendar();
        System.DateTime date = new DateTime(1999, 6, 27, temp_calendar);
        double[] high = new double[]{75.0, 75.25, 75.25, 75.0, 74.125, 74.25};
        double[] low = new double[]{74.125, 74.25, 74.0, 74.5, 73.75, 73.50};
        double[] close = new double[]{75.0, 74.75, 74.25, 74.75, 74.0, 74.0};
        // Create an instance of a HighLowClose Chart
```
HighLowClose hilo = new HighLowClose(axis, date, high, low, close);

    // Set the HighLowClose Chart Title
    chart.ChartTitle.SetTitle("A Simple HighLowClose Chart");

    // Configure the x-axis
    hilo.SetDateAxis("d");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new HiLoEx1());
}
Candlestick Class

```csharp
public class Imsl.Chart2D.Candlestick : HighLowClose

Candlestick plot of stock data.

Two nodes are created as children of this node. One for the up days and one for the down days.
```
Properties

Down
virtual public Imsl.Chart2D.CandlestickItem Down {get; }

Description
The down days of this Candlestick.

Property Value
A CandlestickItem which contains the “Down” attribute value.

Up
virtual public Imsl.Chart2D.CandlestickItem Up {get; }

Description
The up days of this Candlestick.

Property Value
A CandlestickItem which contains the “Up” attribute value.

Constructors

Candlestick
public Candlestick(Imsl.Chart2D.AxisXY axis, System.DateTime start, double[] high, double[] low, double[] close, double[] open)

Description
Constructs a candlestick chart node beginning with specified start date.

Parameters
  axis – An AxisXY which is the parent of this node.
  start – A DateTime that specifies the first date.
  high – A double[] which contains the stock’s high prices.
  low – A double[] which contains the stock’s low prices.
  close – A double[] which contains the stock’s closing prices.
  open – A double[] which contains the stock’s opening prices.

Remarks
Each of the arguments are use to set the related attribute (e.g. “High”, “Low”, “Close” and “Open”).

Candlestick
public Candlestick(Imsl.Chart2D.AxisXY axis, double[] x, double[] high, double[] low, double[] close, double[] open)
Description
Constructs a candlestick chart node beginning with specified axis points.

Parameters
- axis – An AxisXY which is the parent of this node.
- x – A double[] which contains the axis points.
- high – A double[] which contains the stock's high prices.
- low – A double[] which contains the stock's low prices.
- close – A double[] which contains the stock's closing prices.
- open – A double[] which contains the stock's opening prices.

Remarks
Each of the arguments are use to set the related attribute (e.g. “X”, “High”, “Low”, “Close” and “Open”).

Method

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
- draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

CandlestickItem Class

public class Imsl.Chart2D.CandlestickItem : Data
A candlestick for the up days or the down days.
CandlestickItems are created by Candlestick; one for up days and one for down days.

See Also
Imsl.Chart2D.Candlestick (p. 1458)
Method

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SplineData Class

public class Imsl.Chart2D.SplineData : Data
A data set created from a Spline.

See Also
Imsl.Math.Spline (p. 143)

Constructor

SplineData
public SplineData(Imsl.Chart2D.ChartNode parent, Imsl.Math.Spline spline)

Description
Creates a data node from Spline values.

Parameters
parent – A ChartNode which specifies the parent of this data node.
spline – A Spline which specifies the data to be plotted.
Example: SplineData Chart

This example makes use of the SplineData class as well as the two spline smoothing classes in
Imsl.Math.

using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using Imsl.Math;
using Imsl.Chart2D;
using Random = Imsl.Stat.Random;

public class SplineDataEx1 : FrameChart
{
    private const int nData = 21;
    private const int nSpline = 100;

    public SplineDataEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        chart.ChartTitle.SetTitle(new Text("Smoothed Spline"));

        Legend legend = chart.Legend;
        legend.SetTitle(new Text("Legend"));
        legend.SetViewport(0.7, 0.9, 0.1, 0.3);
        legend.IsVisible = true;

        // Original data
        double[] xData = grid(nData);
        double[] yData = new double[nData];
        for (int k = 0; k < nData; k++)
        {
            yData[k] = f(xData[k]);
        }

        Data data = new Data(axis, xData, yData);
        data.DataType = Imsl.Chart2D.Data.DATA_TYPE_MARKER;
        data.MarkerType = Data.MARKER_TYPE_HOLLOW_CIRCLE;
        data.SetTitle("Original Data");

        // Noisy data
        Random random = new Random(123457);
        double[] yNoisy = new double[nData];
        for (int k = 0; k < nData; k++)
        {
            yNoisy[k] = yData[k] + (2.0 * random.NextDouble() - 1.0);
        }
    
    // SplineData IMSL C# Numerical Library
data = new Data(axis, xData, yNoisy);
data.DataType = Imsl.Chart2D.Data.DATA_TYPE_MARKER;
data.MarkerType = Data.MARKER_TYPE_FILLED_SQUARE;
data.MarkerSize = 0.75;
data.SetTitle("Noisy Data");

chartSpline(axis, new CsSmooth(xData, yData), System.Drawing.Color.Red, "CsSmooth");
chartSpline(axis, new CsSmoothC2(xData, yData, nData), System.Drawing.Color.Orange, "CsSmoothC2");
}

{
    Data data = new SplineData(axis, spline);
data.DataType = Imsl.Chart2D.Data.DATA_TYPE_LINE;
data.LineColor = color;
data.SetTitle(title);
}

static private double[] grid(int nData)
{
    double[] xData = new double[nData];
    for (int k = 0; k < nData; k++)
    {
        xData[k] = 3.0 * k / (double) (nData - 1);
    }
    return xData;
}

static private double f(double x)
{
    return 1.0 / (0.1 + System.Math.Pow(3.0 * (x - 1.0), 4));
}

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new SplineDataEx1());
}
public class Imsl.Chart2D.Bar : Data
A bar chart.
The class Bar has children of class Imsl.Chart2DBarItem (p. 1473). The attribute “BarItem” in class Bar is set to theBarItem array of children.

Bar Class
See Also

Imsl.Chart2D.BarSet (p. 1477), Imsl.Chart2D.BarItem (p. 1473)

Constructors

Bar

public Bar(Imsl.Chart2D.AxisXY axis)

Description

Constructs a bar chart.

Parameter

axis – A AxisXY which is the parent of this node.

Bar

public Bar(Imsl.Chart2D.AxisXY axis, double[] y)

Description

Constructs a simple bar chart using supplied y data.

Parameters

axis – A AxisXY which is the parent of this node.

y – A double[] which contains the y data for the simple bar chart

Bar

public Bar(Imsl.Chart2D.AxisXY axis, double[] x, double[] y)

Description

Constructs a simple bar chart using supplied x and y data.

Parameters

axis – A AxisXY which is the parent of this node.

x – A double[] which contains the x data for the simple bar chart.

y – A double[] which contains the y data for the simple bar chart.

Bar

public Bar(Imsl.Chart2D.AxisXY axis, double[][] y)

Description

Constructs a grouped bar chart using supplied x and y data.
Parameters

axis – A AxisXY which is the parent of this node.
y – A double[] which contains the y data for the grouped bar chart. The first index refers to the group and the second refers to the x position.

Bar

public Bar(Imsl.Chart2D.AxisXY axis, double[] x, double[][] y)

Description

Constructs a grouped bar chart using supplied x and y data.

Parameters

axis – A AxisXY which is the parent of this node.
x – A double[] which contains the x data for the grouped bar chart.
y – A double[] which contains the y data for the grouped bar chart. The first index refers to the group and the second refers to the x position.

Bar

public Bar(Imsl.Chart2D.AxisXY axis, double[][] y)

Description

Constructs a stacked, grouped bar chart using supplied y data.

Parameters

axis – A AxisXY which is the parent of this node.
y – A double[] which contains the y data for the stacked, grouped bar chart. The first index refers to the stack, the second refers to the group and the third refers to the x position.

Bar

public Bar(Imsl.Chart2D.AxisXY axis, double[] x, double[][] y)

Description

Constructs a stacked, grouped bar chart using supplied x and y data.

Parameters

axis – A AxisXY which is the parent of this node.
x – A double[] which contains the x data for the stacked, grouped bar chart.
y – A double[] which contains the y data for the stacked, grouped bar chart. The first index refers to the “stack”, the second refers to the group and the third refers to the x position.

Methods

GetBarData

virtual public double[][][] GetBarData()
Description
Returns the “BarData” attribute value.

Returns
A double[][][] that contains the “BarData” attribute value.

Remarks
The value is an array of object that make up a bar chart. The first index refers to the “stack”, the second
refers to the group and the third refers to the x position.

GetBarSet
virtual public Imsl.Chart2D.BarSet[][] GetBarSet()

Description
Returns the BarSet object.

Returns
A BarSet[][] containing the “BarSet” attribute value.

GetBarSet
virtual public Imsl.Chart2D.BarSet GetBarSet(int group)

Description
Returns the BarSet object.

Parameter
    group – An int which specifies the group index.

Returns
A BarSet[][] containing the “BarSet” attribute value.

Remarks
The group index is assumed to be zero. This method is most useful for charts with only a single group.

GetBarSet
virtual public Imsl.Chart2D.BarSet GetBarSet(int stack, int group)

Description
Returns the BarSet object.

Parameters
    stack – An int which specifies the stack index.
    group – An int which specifies the group index.

Returns
A BarSet[][] containing the “BarSet” attribute value.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)
**Description**

Paints this node and all of its children.

**Parameter**

- draw – A Draw which is to be painted.

**Remarks**

This is normally called only by the Paint method in this node’s parent.

**SetBarData**

```csharp
virtual public void SetBarData(double[][][] bardata)
```

**Description**

Sets the “BarData” attribute value.

**Parameter**

- bardata – A double[][][] that specifies the “BarData” attribute value.

**Remarks**

The value is an array of object that make up a bar chart. The first index refers to the “stack”, the second refers to the group and the third refers to the x position.

**SetDataRange**

```csharp
override public void SetDataRange(double[] range)
```

**Description**

Update the data range, `range = {xmin,xmax,ymin,ymax}`

**Parameter**

- range – a double array which contains the updated range, `{xmin,xmax,ymin,ymax}`

**Remarks**

The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

**SetLabels**

```csharp
virtual public void SetLabels(string[] labels)
```

**Description**

Sets up an axis with bar labels.

**Parameter**

- labels – A String[] array with which to label the axis.
Remarks
This turns off the tick marks and sets the “BarType” attribute. It also turns off autoscaling for the axis and sets its “Window” and “Number” and “Ticks” attribute as appropriate for a labeled bar chart. The existing value of the “BarType” attribute is used to determine the axis to be modified.
The number of labels must equal the number of items.

SetLabels
virtual public void SetLabels(string[] labels, int type)

Description
Sets up an axis with bar labels.

Parameters
- labels – A String[] which specifies axis labels.
- type – An int which specifies the “BarType”.

Remarks
This turns off the tick marks and sets the “BarType” attribute. It also turns off autoscaling for the axis and sets its “Window”, “Number” and “Ticks” attributes as appropriate for a labeled bar chart.
The number of labels must equal the number of items.
The bar type determines the axis to be modified. Legal values are:

- Imsl.Chart2D.ChartNode.BAR_TYPE_VERTICAL (p. 1311)
- Imsl.Chart2D.ChartNode.BAR_TYPE_HORIZONTAL (p. 1311)

Example: Stacked Bar Chart
A stacked bar chart is constructed in this example. Bar labels and colors are set and axis labels are set.

using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class BarEx1 : FrameChart
{
    public BarEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        int nStacks = 2;
        int nGroups = 3;
        int nItems = 6;

        // Generate some random data
        double[] dbl = new double[50];
dbl[0]=0.41312962995625035;
dbl[1]=0.15995876895053263;
dbl[2]=0.822528716547005;
dbl[3]=0.4879436763379836;
dbl[4]=0.44364905186692527;
dbl[5]=0.20896329070872555;
dbl[6]=0.988708342522812;
dbl[7]=0.478176623804778;
dbl[8]=0.964768112234352;
dbl[9]=0.6732389937186418;
dbl[10]=0.5668831243079411;
dbl[11]=0.33081942994459734;
dbl[12]=0.27386697614898103;
dbl[13]=0.10880787186704965;
dbl[14]=0.8805853693809824;
dbl[15]=0.901138442534768;
dbl[16]=0.7180829622748057;
dbl[17]=0.4872365638326413;
dbl[18]=0.615607537410854;
dbl[19]=0.1015355280528812;
dbl[20]=0.3158193853638753;
dbl[21]=0.958058275075961;
dbl[22]=0.10778543304578747;
dbl[23]=0.01182928759960888;
dbl[24]=0.09275375134616693;
dbl[25]=0.4859902873228249;
dbl[26]=0.9817642781628322;
dbl[27]=0.5505301300240635;
dbl[28]=0.467363186309925;
dbl[29]=0.1865244274911184;
dbl[30]=0.906980293517674;
dbl[31]=0.927236533193322;
dbl[32]=0.31440695305815347;
dbl[33]=0.4215880116306273;
dbl[34]=0.9991560762956562;
dbl[35]=0.03826317648903991;
dbl[36]=0.785150345014761;
dbl[37]=0.6451521871931544;
dbl[38]=0.7930129038729785;
dbl[39]=0.819301055474365;
dbl[40]=0.5695413465811706;
dbl[41]=0.039285689951912395;
dbl[42]=0.7625752595574732;
dbl[43]=0.31325564481720314;
dbl[44]=0.0482465474704169;
dbl[45]=0.6272275622766595;
dbl[46]=0.09904819350827354;
dbl[47]=0.8934533907287561;
dbl[48]=0.7013979421419555;
dbl[49]=0.5212913217641422;

int z=0;

double[] x = new double[nItems];
double[][][] y = new double[nStacks][][];
for (int i = 0; i < nStacks; i++)
```csharp
// Create an instance of a Bar Chart
Bar bar = new Bar(axis, y);
// Set the Bar Chart Title
chart.ChartTitle.SetTitle("Sales by Region");
// Set the fill outline type;
bar.FillOutlineType = Bar.FILL_TYPE_SOLID;
System.Drawing.Color GREEN = System.Drawing.Color.FromArgb(0, 255, 0);
// Set the Bar Item fill colors
bar.GetBarSet(0, 0).FillColor = System.Drawing.Color.Red;
bar.GetBarSet(0, 1).FillColor = System.Drawing.Color.Yellow;
bar.GetBarSet(0, 2).FillColor = GREEN;
bar.GetBarSet(1, 2).FillColor = System.Drawing.Color.Magenta;
chart.Legend.IsVisible = true;
bar.GetBarSet(0, 0).SetTitle("Red");
bar.GetBarSet(0, 1).SetTitle("Yellow");
bar.GetBarSet(0, 2).SetTitle("Green");
bar.GetBarSet(1, 0).SetTitle("Blue");
bar.GetBarSet(1, 1).SetTitle("Cyan");
bar.GetBarSet(1, 2).SetTitle("Magenta");
// Setup the vertical axis for a labeled bar chart.
bar.SetLabels(labels, Imsl.Chart2D.Bar.BAR_TYPE_VERTICAL);
// Set the text angle
axis.AxisX.AxisLabel.TextAngle = 270;
// Set the Y axis title
```
```csharp
public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new BarEx1());
}
```

Output

```
<table>
<thead>
<tr>
<th>Sales ($million) by widget color</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
</tr>
<tr>
<td>Texas</td>
</tr>
<tr>
<td>Northern California</td>
</tr>
<tr>
<td>Southern California</td>
</tr>
<tr>
<td>Colorado</td>
</tr>
<tr>
<td>New Jersey</td>
</tr>
</tbody>
</table>
```

Sales by Region

- Red
- Yellow
- Green
- Blue
- Cyan
- Magenta
BarItem Class

public class Imsl.Chart2DBarItem : Data
A single bar in a bar chart.

See Also
Imsl.Chart2D.Bar (p. 1464), Imsl.Chart2D.BarSet (p. 1477)

Methods

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node's parent.

SetDataRange
override public void SetDataRange(double[] range)

Description
Update the data range, range = {xmin,xmax,ymin,ymax}

Parameter
range – a double array which contains the updated range, {xmin,xmax,ymin,ymax}

Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.
Example: Stacked Bar Chart

A stacked bar chart is constructed in this example. Bar labels and colors are set and axis labels are set.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class BarEx1 : FrameChart
{
    public BarEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        int nStacks = 2;
        int nGroups = 3;
        int nItems = 6;

        // Generate some random data

        double[] dbl = new double[50];
        dbl[0]=0.41312962995625035;
        dbl[1]=0.15995876895053263;
        dbl[2]=0.8225528716547005;
        dbl[3]=0.48794367683379836;
        dbl[4]=0.44364905186692527;
        dbl[5]=0.20896329070872555;
        dbl[6]=0.9887088342522812;
        dbl[7]=0.4781765623804778;
        dbl[8]=0.9647868112234352;
        dbl[9]=0.673389937186418;
        dbl[10]=0.5668831243079411;
        dbl[11]=0.33081942994459734;
        dbl[12]=0.2738697614898103;
        dbl[13]=0.10880787186704965;
        dbl[14]=0.8805853693809824;
        dbl[15]=0.901138442534768;
        dbl[16]=0.718029622748057;
        dbl[17]=0.48723665383264413;
        dbl[18]=0.6153607537410654;
        dbl[19]=0.10153552805288812;
        dbl[20]=0.3158193853638753;
        dbl[21]=0.9558058275075961;
        dbl[22]=0.10778543304578747;
        dbl[23]=0.011829287599608884;
        dbl[24]=0.09275375134615693;
        dbl[25]=0.4859902873228249;
        dbl[26]=0.9817642781628322;
        dbl[27]=0.5505301300240635;
        dbl[28]=0.467363186309925;
        dbl[29]=0.18652444274911184;
        dbl[30]=0.9066980293517674;
        dbl[31]=0.9272326533193322;
```
int z = 0;

double[] x = new double[nItems];
double[][][] y = new double[nStacks][][];
for (int i = 0; i < nStacks; i++)
{
    y[i] = new double[nGroups][];
    for (int i2 = 0; i2 < nGroups; i2++)
    {
        y[i][i2] = new double[nItems];
    }
}

double dx = 0.5 * System.Math.PI / (x.Length - 1);
for (int istack = 0; istack < y.Length; istack++)
{
    for (int jgroup = 0; jgroup < y[istack].Length; jgroup++)
    {
        for (int kitem = 0; kitem < y[istack][jgroup].Length; kitem++)
        {
            y[istack][jgroup][kitem] = dbl[z]; // r.NextDouble();
            z++;
        }
    }
}

// Create an instance of a Bar Chart
Bar bar = new Bar(axis, y);

// Set the Bar Chart Title
chart.ChartTitle.SetTitle("Sales by Region");

// Set the fill outline type;
bar.FillOutlineType = Bar.FILL_TYPE_SOLID;

System.Drawing.Color GREEN = System.Drawing.Color.FromArgb(0, 255, 0);
// Set the Bar Item fill colors
bar.GetBarSet(0, 0).FillColor = System.Drawing.Color.Red;
```csharp
bar.GetBarSet(0, 1).FillColor = System.Drawing.Color.Yellow;
bar.GetBarSet(0, 2).FillColor = GREEN;
bar.GetBarSet(1, 2).FillColor = System.Drawing.Color.Magenta;

chart.Legend.IsVisible = true;
bar.GetBarSet(0, 0).SetTitle("Red");
bar.GetBarSet(0, 1).SetTitle("Yellow");
bar.GetBarSet(0, 2).SetTitle("Green");
bar.GetBarSet(1, 0).SetTitle("Blue");
bar.GetBarSet(1, 1).SetTitle("Cyan");
bar.GetBarSet(1, 2).SetTitle("Magenta");

// Setup the vertical axis for a labeled bar chart.
bar.SetLabels(labels, Imsl.Chart2D.Bar.BAR_TYPE_VERTICAL);

// Set the text angle
axis.AxisX.AxisLabel.TextAngle = 270;

// Set the Y axis title
axis.AxisY.AxisTitle.SetTitle("Sales ($million) by widget color");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new BarEx1());
}
```
BarSet Class

public class Imsl.Chart2D.BarSet : ChartNode

A set of bars in a bar chart.

A BarSet is created by Imsl.Chart2D.Bar (p. 1464) and contains a collection of Imsl.Chart2DBarItem (p. 1473). Bar creates a BarSet for each stack-group combination. Each BarSet contains the BarItems for that combination. Normally all of the BarItems in a BarSet have the same color, title, etc.
Methods

GetBarItem
virtual public Imsl.Chart2D.BarItem[] GetBarItem()

Description
Returns an array of BarItems.

Returns
A BarItem[] that contains theBarItem attribute value.

Remarks
This is the collection of allBarItem contained in this bar group.

GetBarItem
virtual public Imsl.Chart2D.BarItem GetBarItem(int index)

Description
Returns theBarItem given the index.

Parameter
index – An int which specifies the index.

Returns
A BarItem associated with the specified index.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetDataRange
virtual public void SetDataRange(double[] range)

Description
Update the data range, range = {xmin,xmax,ymin,ymax}

Parameter
range – a double array which contains the updated range, {xmin,xmax,ymin,ymax}
Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

Pie Class

public class Imsl.Chart2D.Pie : Axis
A pie chart.
The angle of the first slice is determined by the attribute “Reference”.
Pie is derived from Axis, because it defines its own mapping to device space.

Constructors

Pie
public Pie(Imsl.Chart2D.Chart chart)

Description
Constructs a Pie chart object.

Parameter
chart – A Chart which specifies the parent of this node.

Remarks
The “Viewport” attribute for this node is set to [0.2,0.8] by [0.2,0.8].

Pie
public Pie(Imsl.Chart2D.Chart chart, double[] y)

Description
Constructs a Pie chart object with a specified number of slices.

Parameters
chart – A Chart which specifies the parent of this node.
y – A double[] which contains the values for the pie chart.

Remarks
An array of y.length Imsl.Chart2D.PieSlice (p. 1483) nodes are created as children of this node and this array is used to define the attribute “PieSlice” in this node.
The “Viewport” attribute for this node is set to [0.2,0.8] by [0.2,0.8].
Methods

**GetPieSlice**

```csharp
virtual public Imsl.Chart2D.PieSlice[] GetPieSlice()
```

**Description**

Returns the PieSlice objects.

**Returns**

A PieSlice[] containing the pie slices to be associated with this node.

**GetPieSlice**

```csharp
virtual public Imsl.Chart2D.PieSlice GetPieSlice(int index)
```

**Description**

Returns a specified PieSlice.

**Parameter**

- `index` – An int specifying the pie slice to return.

**Returns**

A PieSlice which contains the specified slice.

**Remarks**

The “PieSlice” attribute is a 0 based index array.

**MapDeviceToUser**

```csharp
override public void MapDeviceToUser(int devX, int devY, double[] userXY)
```

**Description**

Maps the device coordinates devXY to user coordinates (userX,userY).

**Parameters**

- `devX` – An int which specifies the device x-coordinate.
- `devY` – An int which specifies the device y-coordinate.
- `userXY` – An int[2] in which the the user coordinates are returned.

**MapUserToDevice**

```csharp
override public void MapUserToDevice(double userX, double userY, int[] devXY)
```

**Description**

Maps the user coordinates (userX,userY) to the device coordinates devXY.
Parameters

userX – A double which specifies the user x-coordinate.
userY – A double which specifies the user y-coordinate.
devXY – An int[2] in which the device coordinates are returned.

SetData

virtual public Imsl.Chart2D.PieSlice[] SetData(double[] y)

Description
Changes the data in a Pie chart object.

Parameter

y – A double[] which contains the values for the pie chart.

Returns
A PieSlice[] array containing the updated PieSlice.

Remarks
If the number of slices is unchanged then the existing pie slice array, defined by the attribute “PieSlice” in this node, is reused. If the number is different, a new array is allocated, using the existing PieSlice elements to initialize the new array.

SetUpMapping

override public void SetUpMapping()

Description
Initializes the mappings between user and coordinate space.

Remarks
This must be called whenever the screen size, the window or the viewport may have changed. Generally, it is safest to call this each time the chart is repainted.

Example: Pie Chart

A simple Pie chart is constructed in this example. Pie slice labels and colors are set and one pie slice is exploded from the center. This class extends FrameChart, which manages the window.

using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class PieEx1 : FrameChart
{
    public PieEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);
// Create an instance of a Pie Chart
double[] y = new double[]{10.0, 20.0, 30.0, 40.0};
Pie pie = new Pie(chart, y);

// Set the Pie Chart Title
chart.ChartTitle.SetTitle("A Simple Pie Chart");

// Set the colors of the Pie Slices
PieSlice[] slice = pie.GetPieSlice();
slice[0].FillColor = System.Drawing.Color.Red;

// Set the Pie Slice Labels
pie.LabelType = Imsl.Chart2D.Pie.LABEL_TYPE_TITLE;
slice[0].SetTitle("Fish");
slice[1].SetTitle("Pork");
slice[2].SetTitle("Poultry");
slice[3].SetTitle("Beef");

// Explode a Pie Slice
slice[0].Explode = 0.2;

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new PieEx1());
}
}
PieSlice Class

public class Imsl.Chart2D.PieSlice : Data

One wedge of a pie chart.

Imsl.Chart2D.Pie (p. 1479) creates PieSlice objects as its children, one per pie wedge. A specific slice can be retrieved using the method Imsl.Chart2D.Pie.GetPieSlice(System.Int32) (p. 1480). All of the slices can be retrieved using the method Imsl.Chart2D.Pie.GetPieSlice (p. 1480).
The drawing of the slice is controlled by the fill attributes (specified with FillType (p. 1322)) in this node.

**Methods**

**Paint**

```csharp
override public void Paint(Imsl.Chart2D.Draw draw)
```

**Description**

Paints this node and all of its children.

**Parameter**

- `draw` – A Draw which is to be painted.

**Remarks**

This is normally called only by the Paint method in this node’s parent.

**SetAngles**

```csharp
virtual protected internal void SetAngles(double angleA, double angleB)
```

**Description**

Sets the angles, in degrees, that determine the extent of this slice.

**Parameters**

- `angleA` – A double that specifies the angle, in degrees, at which the slice begins.
- `angleB` – A double that specifies the angle, in degrees, at which the slice ends.

---

**Dendrogram Class**

```csharp
public class Imsl.Chart2D.Dendrogram : Data
```

A Dendrogram chart for cluster analysis.

**Properties**

**Coordinates**

```csharp
virtual public double[,] Coordinates {get; set; }
```
Description
The cluster coordinates in the Dendrogram object.

Property Value
An double[][] containing the “Coordinates” attribute value.

LeftSons
virtual public int[] LeftSons {get; set; }

Description
The left sons of each merged cluster.

Property Value
An int[] containing the “LeftSons” attribute value.

Levels
virtual public double[] Levels {get; set; }

Description
Specifies the levels at which the clusters are joined.

Property Value
A double[] containing the “Levels” attribute value.

Order
virtual public int[] Order {get; set; }

Description
The cluster order in the Dendrogram object.

Property Value
An int[] containing the “Order” attribute value.

RightSons
virtual public int[] RightSons {get; set; }

Description
The right sons of each merged cluster.

Property Value
An int[] containing the “RightSons” attribute value.

Constructors

Dendrogram
**Description**
Constructs a vertical Dendrogram chart using a supplied ClusterHierarchical object.

**Parameters**
- axis – An AxisXY specifying the parent of this node.
- clusterHierarchical – A ClusterHierarchical used as a source object for the Dendrogram.

```csharp
Dendrogram(Imsl.Chart2D.AxisXY axis, double[] clusterLevel, int[] leftSons, int[] rightSons)
```

**Description**
Constructs a vertical Dendrogram chart using supplied data.

**Parameters**
- axis – An AxisXY specifying the parent of this node.
- clusterLevel – A double[] which contains the levels at which the clusters are joined.
- leftSons – An int[] which contains the left sons of each merged cluster.
- rightSons – An int[] which contains the right sons of each merged cluster.

```csharp
```

**Description**
Constructs a Dendrogram chart using a supplied ClusterHierarchical object.

**Parameters**
- axis – An AxisXY specifying the parent of this node.
- clusterHierarchical – A ClusterHierarchical object used as a source for the Dendrogram.
- type – An int which specifies the Dendrogram type.

**Remarks**
The types possible types of Dendrograms are DENDROGRAM_TYPE_VERTICAL (p. 1313) and DENDROGRAM_TYPE_HORIZONTAL (p. 1313).

```csharp
Dendrogram(Imsl.Chart2D.AxisXY axis, double[] clusterLevel, int[] leftSons, int[] rightSons, int type)
```

**Description**
Constructs a Dendrogram chart using supplied data.
Parameters

- **axis** – An AxisXY specifying the parent of this node.
- **clusterLevel** – A double[] which contains the levels at which the clusters are joined.
- **leftSons** – An int[] which contains the left sons of each merged cluster.
- **rightSons** – An int[] which contains the right sons of each merged cluster.
- **type** – An int which specifies the Dendrogram type.

Remarks

The types possible types of Dendrograms are DENDROGRAM_TYPE_VERTICAL (p. 1313) and DENDROGRAM_TYPE_HORIZONTAL (p. 1313).

Methods

**Paint**

*override public void Paint(Imsl.Chart2D.Draw draw)*

Description

Paints this node and all of its children.

Parameter

- **draw** – A Draw which is to be painted.

Remarks

This is normally called only by the Paint method in this node’s parent.

**SetDataRange**

*override public void SetDataRange(double[] range)*

Description

Update the data range, range = \{xmin, xmax, ymin, ymax\}

Parameter

- **range** – a double array which contains the updated range, \{xmin, xmax, ymin, ymax\}

Remarks

The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

**SetLabels**

*virtual public void SetLabels(string[] labels)*

Description

Sets up the axis labels for Dendrogram plot.
Parameter

- labels – A String[] containing the axis labels.

Remarks

The number of labels must equal the number of items.
This method turns off autoscaling on the axis and sets the Window attribute depending on the number of points being plotted.
Note that user-defined labels will be re-ordered to match the order of the clusters displayed in the plot.

SetLineColors

virtual public void SetLineColors(System.Drawing.Color[] colors)

Description

Define colors for individual clusters.

Parameter

- colors – A Color[] which contains each color to use for the subclusters.

Remarks

The color of the top most level should be set using Dendrogram.LineColor. This property will color N clusters, where N is the number of elements in colors.

Example: Dendrogram

A Dendrogram.

```csharp
using Imsl.Chart2D;
using Imsl.Stat;
using System;
using System.Windows.Forms;
using System.Drawing;

public class DendrogramEx1 : FrameChart
{
    public DendrogramEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);
        double[,] data = {{.38, 626.5, 601.3, 605.3},
                         {.18, 654.0, 647.1, 641.8},
                         {.07, 677.2, 676.5, 670.5},
                         {.09, 639.9, 640.3, 636.0},
                         {.19, 614.7, 617.3, 606.2},
                         {.12, 670.2, 666.0, 659.3},
                         {.20, 651.1, 645.2, 643.4},
                         {.41, 645.4, 645.8, 644.8},
                         {.07, 683.5, 682.9, 674.3},
                         {.39, 648.6, 647.8, 643.1},
                         {.21, 650.4, 650.8, 643.9},
                         {.24, 637.0, 636.9, 626.5},
```

Dissimilarities dist = new Dissimilarities(data, 0, 1, 1);
double[,] distanceMatrix = dist.DistanceMatrix;
ClusterHierarchical clink = new ClusterHierarchical(
    dist.DistanceMatrix, 4, 0);

int nClusters = 4;
int[] iclus = clink.GetClusterMembership(nClusters);
int[] nclus = clink.GetObsPerCluster(nClusters);

// use either method below to create the chart
Dendrogram dc = new Dendrogram(axis, clink,
    Data.DENDROGRAM_TYPE_HORIZONTAL);

dc.SetLabels(lab);
dc.SetLineColors(new Color[] {Color.Blue, Color.Green, Color.Red,
    Color.Orange});

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new DendrogramEx1());
}

}
Example: Dendrogram

A Dendrogram.

```csharp
using Imsl.Chart2D;
using Imsl.Stat;
using System;
using System.Windows.Forms;
using System.Drawing;

public class DendrogramEx2 : FrameChart
{
    public DendrogramEx2()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);
```
```plaintext
double[,] data = {{5.1, 3.5, 1.4, .2},
                 {4.9, 3.0, 1.4, .2},
                 {4.7, 3.2, 1.3, .2},
                 {4.6, 3.1, 1.5, .2},
                 {5.0, 3.6, 1.4, .2},
                 {5.4, 3.9, 1.7, .4},
                 {4.6, 3.4, 1.4, .3},
                 {5.0, 3.4, 1.5, .2},
                 {4.4, 2.9, 1.4, .2},
                 {4.9, 3.1, 1.5, .1},
                 {5.4, 3.7, 1.5, .2},
                 {4.8, 3.4, 1.6, .2},
                 {4.8, 3.0, 1.4, .1},
                 {4.3, 3.0, 1.1, .1},
                 {5.8, 4.0, 1.2, .2},
                 {5.7, 4.4, 1.5, .4},
                 {5.4, 3.9, 1.3, .4},
                 {5.1, 3.5, 1.4, .3},
                 {5.7, 3.8, 1.7, .3},
                 {5.1, 3.8, 1.5, .3},
                 {5.4, 3.4, 1.7, .2},
                 {5.1, 3.7, 1.5, .4},
                 {4.6, 3.6, 1.0, .2},
                 {5.1, 3.3, 1.7, .5},
                 {4.8, 3.4, 1.9, .2},
                 {5.0, 3.0, 1.6, .2},
                 {5.0, 3.4, 1.6, .4},
                 {5.2, 3.5, 1.5, .2},
                 {5.2, 3.4, 1.4, .2},
                 {4.7, 3.2, 1.6, .2},
                 {4.8, 3.1, 1.6, .2},
                 {5.4, 3.4, 1.5, .4},
                 {5.2, 4.1, 1.5, .1},
                 {5.5, 4.2, 1.4, .2},
                 {4.9, 3.1, 1.5, .2},
                 {5.0, 3.2, 1.2, .2},
                 {5.5, 3.5, 1.3, .2},
                 {4.9, 3.6, 1.4, .1},
                 {4.4, 3.0, 1.3, .2},
                 {5.1, 3.4, 1.5, .2},
                 {5.0, 3.5, 1.3, .3},
                 {4.5, 2.3, 1.3, .3},
                 {4.4, 3.2, 1.3, .2},
                 {5.0, 3.5, 1.6, .6},
                 {5.1, 3.8, 1.9, .4},
                 {4.8, 3.0, 1.4, .3},
                 {5.1, 3.8, 1.6, .2},
                 {4.6, 3.2, 1.4, .2},
                 {5.3, 3.7, 1.5, .2},
                 {5.0, 3.3, 1.4, .2},
                 {7.0, 3.2, 4.7, 1.4},
                 {6.4, 3.2, 4.5, 1.5},
                 {6.9, 3.1, 4.9, 1.5},
                 {5.5, 2.3, 4.0, 1.3},
                 {6.5, 2.8, 4.6, 1.5},
                 {7.0, 3.2, 4.7, 1.4},
                 {6.4, 3.2, 4.5, 1.5},
                 {6.9, 3.1, 4.9, 1.5},
                 {5.5, 2.3, 4.0, 1.3},
                 {6.5, 2.8, 4.6, 1.5}}
```
Dissimilarities dist = new Dissimilarities(data, 0, 1, 1);
double[,] distanceMatrix = dist.DistanceMatrix;
ClusterHierarchical clink = new ClusterHierarchical(dist.DistanceMatrix, 2, 0);

int nClusters = 4;
it[] iclus = clink.GetClusterMembership(nClusters);
it[] nclus = clink.GetObsPerCluster(nClusters);

// use either method below to create the chart
// Dendrogram dc = new Dendrogram(axis, clink);
Dendrogram dc = new Dendrogram(axis, clink.ClusterLevel, clink.ClusterLeftSons, clink.ClusterRightSons);

// set colors
public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new DendrogramEx2());
}

Output
Polar Class

public class Imsl.Chart2D.Polar : Axis
This Axis node is used for polar charts.
In a polar plot, the (x,y) coordinates in Imsl.Chart2D.Data (p. 1372) nodes are interpreted as (r,theta) values. The “Viewport” attribute for this node is set to [0.1,0.9] by [0.1,0.9].

Properties

AxisR
virtual public Imsl.Chart2D.AxisR AxisR {get; }

Description
Return the radius axis node.

AxisTheta
virtual public Imsl.Chart2D.AxisTheta AxisTheta {get; }

Description
Returns the angular axis node.

GridPolar
virtual public Imsl.Chart2D.GridPolar GridPolar {get; }

Description
A grid for the polar plot.

Constructor

Polar
public Polar(Imsl.Chart2D.Chart chart)

Description
Creates a Polar object.

Parameter
chart – A Chart which specifies the parent of this node.
Methods

MapDeviceToUser
override public void MapDeviceToUser(int devX, int devY, double[] userRT)

Description
Map the device coordinates to polar coordinates.

Parameters
- devX – An int which specifies the device x-coordinate.
- devY – An int which specifies the device y-coordinate.
- userRT – A double[2] in which the user coordinates, (radius,theta), are returned.

MapUserToDevice
override public void MapUserToDevice(double userRadius, double userTheta, int[] devXY)

Description
Map the polar coordinates (userRadius, userAngle) to the device coordinates devXY.

Parameters
- userRadius – A double which specifies the user radius coordinate.
- userTheta – A double which specifies the user angle coordinate.
- devXY – An int[2] in which the device coordinates are returned.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
- draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetUpMapping
override public void SetUpMapping()

Description
Initializes the mappings between user and coordinate space.

Remarks
This must be called whenever the screen size, the window or the viewport may have changed.
Heatmap Class

public class Imsl.Chart2D.Heatmap : Data
Heatmap creates a chart from a two-dimensional array of double precision values or Color values. Optionally, each cell in the heatmap can be labeled. If the input is a two-dimensional array of double values then a Colormap object is used to map the real values to colors.

See Also
Imsl.Chart2D.Heatmap.Colormap (p. 1497)

Properties

---

**Colormap**

virtual public Imsl.Chart2D.Colormap Colormap {get; set; }

Description
Specifies the value of the “Colormap” attribute.

Property Value
A Colormap which contains the value of the “Colormap” attribute.

Remarks
This is the Colormap associated with this Heatmap. By default, Colormap = null.

---

**HeatmapLegend**

virtual public Imsl.Chart2D.Heatmap.Legend HeatmapLegend {get; }

Description
Specifies the heatmap legend.

Property Value
A Legend object associated with the Heatmap.

Remarks
By default, the legend is not drawn because the IsVisible (p. 1302) property is set to false. To show the legend set heatmap.HeatmapLegend.IsVisible = true;
Constructors

**Heatmap**

public Heatmap(Imsl.Chart2D.AxisXY axis, double xmin, double xmax, double ymin, double ymax, System.Drawing.Color[,] color)

**Description**

Creates a Heatmap from an array of Color values.

**Parameters**

- **axis** – An AxisXY which contains the parent of this node.
- **xmin** – A double which specifies the minimum x-value of the color data.
- **xmax** – A double which specifies the maximum x-value of the color data.
- **ymin** – A double which specifies the minimum y-value of the color data.
- **ymax** – A double which specifies the maximum y-value of the color data.
- **color** – A Color[,] which specifies the color values.

**Remarks**

The value of color[0,0] is the color of the cell whose lower left corner is (xmin, ymin).

**Heatmap**

public Heatmap(Imsl.Chart2D.AxisXY axis, double xmin, double xmax, double ymin, double ymax, double zmin, double zmax, double[,] data, Imsl.Chart2D.Colormap colormap)

**Description**

Creates a Heatmap from a double[,] and a Colormap.

**Parameters**

- **axis** – An AxisXY object which specifies the parent of this node.
- **xmin** – A double which specifies the minimum x-value of the color data.
- **xmax** – A double which specifies the maximum x-value of the color data.
- **ymin** – A double which specifies the minimum y-value of the color data.
- **ymax** – A double which specifies the maximum y-value of the color data.
- **zmin** – A double which specifies the data value that corresponds to the initial (t=0) value in the Colormap.
- **zmax** – A double which specifies the data value that corresponds to the final (t=1) value in the Colormap.
- **data** – A double[,] containing the data values.
- **colormap** – Maps the values in data to colors.
Remarks
The $x$-interval ($x_{\text{min}}, x_{\text{max}}$) is uniformly divided and mapped into the first index of data. The $y$-interval ($y_{\text{min}}, y_{\text{max}}$) is uniformly divided and mapped into the second index of data. So, the value of 

data[0,0] is used to determine the color of the cell whose lower left corner is ($x_{\text{min}}, y_{\text{min}}$).

If a cell has a data value equal to $t$ then its color is the value of the colormap at $s$, where
\[
s = \frac{t - z_{\text{min}}}{z_{\text{max}} - z_{\text{min}}}
\]

Methods

GetHeatmapLabels
virtual public Imsl.Chart2D.Text[,] GetHeatmapLabels()

Description
Returns the value of the “HeatmapLabels” attribute.

Returns
A Text[,] that contains the values of the “HeatmapLabels” attribute.

Remarks
By default, GetHeatmapLabels = null.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter

draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetDataRange
override public void SetDataRange(double[] range)

Description
Update the data range, range = {$x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}$}

Parameter

range – a double array which contains the updated range, {$x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}$}
Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

SetHeatmapLabels
virtual public void SetHeatmapLabels(string[,] labels)

Description
Sets the value of the “HeatmapLabels” attribute.

Parameter
labels – A string[,] used to create a Text[,] that specifies the Heatmap labels.

Remarks
Each Text object is created from the corresponding label value with TEXT_X_CENTER|TEXT_Y_CENTER alignment.

See Also: Imsl.Chart2D.Text (p. 1385), TEXT_X_CENTER (p. 1317), TEXT_Y_CENTER (p. 1318)

SetHeatmapLabels
virtual public void SetHeatmapLabels(Imsl.Chart2D.Text[,] labels)

Description
Sets the value of the “HeatmapLabels” attribute.

Parameter
labels – A Text[,] that specifies the Heatmap labels.

Remarks
The default alignment for Text is TEXT_X_CENTER|TEXT_Y_CENTER.

See Also: Imsl.Chart2D.Text (p. 1385), TEXT_X_CENTER (p. 1317), TEXT_Y_CENTER (p. 1318)

Example: Heatmap from Color array
A 5 by 10 array of Color objects is created by linearly interpolating red along the x-axis, blue along the y-axis and mixing in a random amount of green. The data range is set to [0,10] by [0,1].

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class HeatmapEx1 : FrameChart
{
    public HeatmapEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);
```
double xmin = 0.0;
double xmax = 10.0;
double ymin = 0.0;
double ymax = 1.0;

int nxRed = 5;
int nyBlue = 10;

System.Random random = new System.Random((System.Int32) 123457L);

int z=0;
int []d=new int[50];
d[0]=34;
d[1]=212;
d[2]=122;
d[3]=86;
d[4]=165;
d[5]=62;
d[6]=195;
d[7]=161;
d[8]=103;
d[9]=155;
d[10]=104;
d[12]=217;
d[13]=252;
d[14]=13;
d[15]=97;
d[16]=104;
d[17]=74;
d[18]=65;
d[19]=248;
d[20]=189;
d[21]=195;
d[22]=105;
d[23]=191;
d[24]=237;
d[25]=28;
d[26]=234;
d[27]=67;
d[28]=172;
d[29]=146;
d[30]=129;
d[31]=2;
d[32]=228;
d[33]=162;
d[34]=25;
d[35]=177;
d[36]=109;
d[37]=251;
d[38]=215;
d[39]=243;
d[40]=106;
d[41]=154;
d[42]=22;
d[43]=65;
d[44]=101;
d[45]=192;
d[46]=103;
d[47]=28;
d[48]=32;
d[49]=143;

for (int i = 0; i < nxRed; i++)
{
    for (int j = 0; j < nyBlue; j++)
    {
        int r = (int) (255.0 * i / nxRed);
        //
        int g =d[z];
        z++;
        int b = (int) (255.0 * j / nyBlue);
        color[i,j] = System.Drawing.Color.FromArgb(r, g, b);
    }
}
Heatmap heatmap = new Heatmap(axis, xmin, xmax, ymin, ymax, color);
axis.AxisX.AxisTitle.SetTitle("Red");
axis.AxisY.AxisTitle.SetTitle("Blue");
}

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new HeatmapEx1());
}
Example: Heatmap from Color array

A 5 by 10 data array is created by linearly interpolating from the lower left corner to the upper right corner and adding in a uniform random variable. A red temperature color map is used. This maps the minimum data value to light green and the maximum data value to dark green.

The legend is enabled by setting its paint attribute to true.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class HeatmapEx2 : FrameChart
{
    public HeatmapEx2()
    {
```
Chart chart = this.Chart;

AxisXY axis = new AxisXY(chart);

int nx = 5;
int ny = 10;
double xmin = 0.0;
double xmax = 10.0;
double ymin = -3.0;
double ymax = 2.0;
double fmax = nx + ny - 1;

double[,] data = new double[nx, ny];

System.Random random = new System.Random((System.Int32)123457L);

double[] dbl = new double[50];
dbl[0] = 0.41312962295625035;
dbl[1] = 0.15995876895053263;
dbl[2] = 0.8225528716547005;
dbl[3] = 0.48794367683739836;
dbl[4] = 0.44364905186692527;
dbl[5] = 0.20896329070872555;
dbl[6] = 0.988708342522812;
dbl[7] = 0.4781765623804778;
dbl[8] = 0.947868112234352;
dbl[9] = 0.6732389937186418;
dbl[10] = 0.566831243079411;
dbl[11] = 0.33081942994459734;
dbl[12] = 0.27386697614898103;
dbl[13] = 0.10880787186704965;
dbl[14] = 0.880583693809824;
dbl[15] = 0.901138442534768;
dbl[16] = 0.7180829622748057;
dbl[17] = 0.48723665383264413;
dbl[18] = 0.6153607537410654;
dbl[19] = 0.10153552805288812;
dbl[20] = 0.3158193853638753;
dbl[21] = 0.9558058275075961;
dbl[22] = 0.10778543304578747;
dbl[23] = 0.01182928759908884;
dbl[24] = 0.09275375134615693;
dbl[25] = 0.4859902873228249;
dbl[26] = 0.9817642781628322;
dbl[27] = 0.5505301300240635;
dbl[28] = 0.467363186309925;
dbl[29] = 0.1865244274911184;
dbl[30] = 0.9066980293517674;
dbl[31] = 0.927236533193322;
dbl[32] = 0.31440695305815347;
dbl[33] = 0.4215880116306273;
dbl[34] = 0.9991560762956562;
dbl[35] = 0.0386317648903991;
dbl[36] = 0.785150345014761;
dbl[37] = 0.645152187931544;
dbl[38] = 0.7930129038729785;
double d[0] = 0.819301055474355;
double d[1] = 0.5695413465811706;
double d[2] = 0.039285689951912395;
double d[3] = 0.7625752595574732;
double d[4] = 0.31325564481720314;
double d[5] = 0.0482465474704169;
double d[6] = 0.6272275622766595;
double d[7] = 0.9904819350827354;
double d[8] = 0.8934533907186641;
double d[9] = 0.7013979421419555;
double d[10] = 0.5212913217641422;

int z = 0;
for (int i = 0; i < nx; i++)
{
    for (int j = 0; j < ny; j++)
    {
        double data[i, j] = i + j + d[z];
        z++;
    }
}

Heatmap heatmap = new Heatmap(axis, xmin, xmax, ymin, ymax, 0.0, fmax,
data, Imsl.Chart2D.Colormap.Fields.RED_TEMPERATURE);
heatmap.HeatmapLegend.IsVisible = true;
heatmap.HeatmapLegend.SetTitle("Heat");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new HeatmapEx2());
}
Example: Heatmap with Labels

A 5 by 10 array of random data is created and a similarly sized array of strings is also created. These labels contain spreadsheet-like indices and the random data value expressed as a percentage.

The legend is enabled by setting its paint attribute to true. The tick marks in the legend are formatted using the percentage NumberFormat object. A title is also set in the legend.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class HeatmapEx3 : FrameChart
{
    public HeatmapEx3()
    {
```
Chart chart = this.Chart;

AxisXY axis = new AxisXY(chart);

double xmin = 0.0;
double xmax = 10.0;
double ymin = 0.0;
double ymax = 1.0;

int nx = 5;
int ny = 10;
double[,] data = new double[nx,ny];

System.String[,] labels = new System.String[nx,ny];
System.Random random = new System.Random((System.Int32) 123457L);

double[] dbl = new double[50];
dbl[0]=0.4132962996625035;
dbl[1]=0.15995876895053263;
dbl[2]=0.8225528716547005;
dbl[3]=0.48794367683379836;
dbl[4]=0.44364905186692527;
dbl[5]=0.20896329070872555;
dbl[6]=0.9887088342522812;
dbl[7]=0.4781765623804778;
dbl[8]=0.9647868112234352;
dbl[9]=0.6732389937186418;
dbl[10]=0.5668831243079411;
dbl[11]=0.33081942994459734;
dbl[12]=0.273886897614898103;
dbl[13]=0.10880787186704965;
dbl[14]=0.88058536935809824;
dbl[15]=0.9011384425347688;
dbl[16]=0.7108029622748057;
dbl[17]=0.48723656383264413;
dbl[18]=0.6153607537410654;
dbl[19]=0.10153552805288812;
dbl[20]=0.3158193853638753;
dbl[21]=0.9558058275075961;
dbl[22]=0.10778543304578747;
dbl[23]=0.011829287599608884;
dbl[24]=0.09275375134615693;
dbl[25]=0.4859902873228249;
dbl[26]=0.9817427816283222;
dbl[27]=0.5505301300240635;
dbl[28]=0.467363186309925;
dbl[29]=0.18652444274911184;
dbl[30]=0.9066980293517674;
dbl[31]=0.9272326533193322;
dbl[32]=0.31440695305815347;
dbl[33]=0.4215880116306273;
dbl[34]=0.9991560762956562;
dbl[35]=0.0386317648903991;
dbl[36]=0.785150345014761;
dbl[37]=0.6451521871931544;
dbl[38]=0.7930129038729785;
dbl[39]=0.819301055474355;
dbl[40]=0.5695413465811706;
dbl[41]=0.0392856481720314;
dbl[42]=0.7625752955477432;
dbl[43]=0.31325564481720314;
dbl[44]=0.0482465474704169;
dbl[45]=0.627275622766595;
dbl[46]=0.09904819350827354;
dbl[47]=0.8934533907186641;
dbl[48]=0.7013979421419555;
dbl[49]=0.5212913217641422;

int z=0;
for (int i = 0; i < nx; i++)
{
    for (int j = 0; j < ny; j++)
    {
        data[i,j] = dbl[z];//random.NextDouble();
        z++; 
        labels[i,j] = "ABCDE"[i] + System.Convert.ToString(j) + "\n" +
            data[i,j].ToString("P0");
    }
}

Heatmap heatmap = new Heatmap(axis, xmin, xmax, ymin, ymax, 0.0, 1.0,
    data, Imsl.Chart2D.Colormap_Fields.BLUE);
heatmap.SetHeatmapLabels(labels);
heatmap.TextColor = System.Drawing.Color.FromName("orange");
heatmap.HeatmapLegend.IsVisible = true;
heatmap.HeatmapLegend.TextFormat = "P0";
heatmap.HeatmapLegend.SetTitle("Percentage");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new HeatmapEx3());
}
}
Heatmap.Legend Class

public class Imsl.Chart2D.Heatmap.Legend : AxisXY

A legend for use with a Heatmap.

This Legend should be used with Heatmaps, rather than the usual Chart legend. The “Viewport” attribute for this node is set to [0.83,0.98] by [0.1,0.6].
Method

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

Treemap Class

public class Imsl.Chart2D.Treemap : Data
Treemap creates a chart from two arrays of double precision values or one data array and one array of Color values. The size of each element is scaled using the first input array. The second array of values or colors is used to shade the corresponding area.


See Also
Imsl.Chart2D.Treemap.Colormap (p. 1510)

Properties

Colormap
virtual public Imsl.Chart2D.Colormap Colormap {get; set; }

Description
Specifies the value of the “Colormap” attribute.

Property Value
A Colormap which contains the value of the “Colormap” attribute.
Default: Colormap = Colormap.Fields.BLUE.
Remarks
This is the colormap associated with this Treemap.

Orientation
virtual public Imsl.Chart2D.Treemap.OrientationMethod Orientation {get; set; }

Description
Specifies the value of the “Orientation” attribute.

Property Value
One of Treemap.OrientationMethod.Automatic, Treemap.OrientationMethod.RowFirst, or Treemap.OrientationMethod.ColumnFirst. The default behavior is TreemapOrientationMethod.Automatic and filling the graph is based on the aspect ratio of the parent Axis object such that if the height is less than the width, columns are drawn first; otherwise rows are drawn first.


Remarks
This is the Orientation associated with this Treemap.

TreemapLegend
virtual public Imsl.Chart2D.Treemap.Legend TreemapLegend {get; }

Description
Specifies the treemap legend.

Property Value
A Legend object associated with the Treemap.

Remarks
Default: The legend is not drawn because the IsVisible (p. 1302) property is set to false. To show the legend set treemap.TreemapLegend.IsVisible = true;

Constructors

Treemap
public Treemap(Imsl.Chart2D.AxisXY axis, double[] data, System.Drawing.Color[] colors)

Description
Constructs a treemap using supplied data and color array.
Parameters
axis – The AxisXY parent of this node.
data – A double array containing the area values for the treemap in decreasing order.
colors – A Color array, one for each area.

Treemap
public Treemap(Imsl.Chart2D.AxisXY axis, double[] data, double[] shades, Imsl.Chart2D.Colormap colormap)

Description
Constructs a treemap using the supplied data and a colormap.

Parameters
axis – The AxisXY parent of this node.
data – A double array containing the area values for the treemap in decreasing order.
shades – An array of double values to use for shading each area.
colormap – A Colormap to use for the shading.

Methods

GetTreemapLabels
virtual public Imsl.Chart2D.Text[] GetTreemapLabels()

Description
Returns the value of the “TreemapLabels” attribute.

Returns
A Text[] that contains the values of the “TreemapLabels” attribute. If the Labels have not been set, this method returns null.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints this node and all of its children.

Parameter
draw – A Draw object which is to be painted.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetDataRange
override public void SetDataRange(double[] range)
Update the data range, range = \{x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}\}

**Parameter**

- range – a double array which contains the updated range, \{x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}\} or \{x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}, z_{\text{min}}, z_{\text{max}}\}

**Remarks**

The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

**SetTreemapLabels**

virtual public void SetTreemapLabels(string[] labels)

**Description**

Sets the value of the “TreemapLabels” attribute.

**Parameter**

- labels – A String[] used to create a Text[] that specifies the Treemap labels.

**Remarks**

Each Text object is created from the corresponding label value with TEXT_X_CENTER|TEXT_Y_CENTER alignment.

Default: No labels are drawn.

See Also: Imsl.Chart2D.Text (p. 1385), TEXT_X_CENTER (p. 1317), TEXT_Y_CENTER (p. 1318)

**SetZRange**

public void SetZRange(double[] range)

**Description**

Set the Z data (shading) range, range = \{z_{\text{min}}, z_{\text{max}}\}.
Parameter
range – A double array which contains the data range, \{zmin, zmax\}.
Default: Values computed from the input shades array or else is \{0, 1\}.

Example: Treemap with sorted data
A treemap is constructed from area and population data of the 15 largest US states. Each rectangle is
proportional to the state’s area (in square miles) and is shaded by its population (in millions).

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class TreemapEx1 : FrameChart
{
    public TreemapEx1()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        double[] areas = {570374, 261914, 155973, 145556, 121364, 113642, 109806, 103729, 97105, 96002, 82751, 82168, 81823, 79617, 76878};

        double[] population = {0.626932, 20.851820, 33.871648, 9.02195, 1.819046, 5.130632, 1.998257, 4.301261, 0.493782, 3.421399, 1.293953, 2.233169, 2.688418, 4.919479, 0.1711263};


        Treemap treemap = new Treemap(axis, areas, population,
            Colormap_Fields.BLUE_WHITE);
        treemap.SetZRange(new double[]{0, 40});
        treemap.SetTreemapLabels(names);
        treemap.TreemapLegend.IsVisible = true;
        treemap.TreemapLegend.SetTitle("Pop. (M)");
        treemap.TreemapLegend.TextFormat = "0";
        axis.SetViewport(0.05, 0.8, 0.1, 0.95);
    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new TreemapEx1());
    }
}
```
**Example: Treemap with unsorted data**

A treemap is constructed from business analytics data. The area is proportional to the company’s sales volume and the color scale maps to a performance indicator. The raw data are unsorted and must be sorted by the area values before creating the chart.

```csharp
using Imsl.Chart2D;
using System;
using System.Windows.Forms;

public class TreemapEx2 : FrameChart
{
    public TreemapEx2()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);
```
double[] areas = {834, 11359, 1621, 12282, 14646, 8686, 12114, 61402, 582, 9448, 1678};

double[] perf = {-1.75, -0.99, -1.4, 0.12, -0.28, -1.14, -0.06, 0.37, -0.66, 0, 0.75};


// sort data
double[] s_perf = new double[perf.Length];
String[] s_lab = new String[perf.Length];
int[] idx = new int[perf.Length];
Imsl.Stat.Sort.Descending(areas, idx);
for (int i = 0; i < perf.Length; i++)
{
    s_perf[i] = perf[idx[i]];
    s_lab[i] = labels[idx[i]];
}

Treemap treemap = new Treemap(axis, areas, s_perf, Colormap.Fields.BLUE_GREEN_RED_YELLOW);
treemap.SetTreemapLabels(s_lab);
treemap.TreemapLegend.IsVisible = true;
treemap.TreemapLegend.SetTitle("Scale");
treemap.TreemapLegend.TextFormat = "0.0";
}

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new TreemapEx2());
}
}
Colormap Interface

public interface Imsl.Chart2D.Colormap

An interface used to implement colormaps.
Method

**GetColor**

```csharp
abstract public System.Drawing.Color GetColor(double t)
```

**Description**
Maps the parameterization interval [0,1] into Colors.

**Parameter**

- **t** – A double in the interval [0,1] to be mapped.

**Returns**

A Color corresponding to t.

---

**Colormap_Fields Structure**

```csharp
public structure Imsl.Chart2D.Colormap_Fields
```

Colormaps are mappings from the unit interval to Colors.
They are a one-dimensional parameterized path through the color cube.

**See Also**

Imsl.Chart2D.Heatmap (p. 1497)

---

**Fields**

**BLUE**

```csharp
public Imsl.Chart2D.Colormap BLUE
```

**Description**

A linear blue colormap.

**BLUE_GREEN_RED_YELLOW**

```csharp
public Imsl.Chart2D.Colormap BLUE_GREEN_RED_YELLOW
```
Description
A blue, green, red and yellow colormap.

BLUE_RED
public Imsl.Chart2D.Colormap BLUE_RED

Description
A linear blue and red colormap.

BLUE_WHITE
public Imsl.Chart2D.Colormap BLUE_WHITE

Description
A linear blue and white colormap.

BW_LINEAR
public Imsl.Chart2D.Colormap BW_LINEAR

Description
A linear black and white (grayscale) colormap.

GREEN
public Imsl.Chart2D.Colormap GREEN

Description
A linear green colormap.

GREEN_PINK
public Imsl.Chart2D.Colormap GREEN_PINK

Description
A linear green and pink colormap.

GREEN_RED_BLUE_WHITE
public Imsl.Chart2D.Colormap GREEN_RED_BLUE_WHITE

Description
A green, red, blue and white colormap.

GREEN_WHITE_EXPONENTIAL
public Imsl.Chart2D.Colormap GREEN_WHITE_EXPONENTIAL

Description
An exponential green and white colormap.

GREEN_WHITE_LINEAR
public Imsl.Chart2D.Colormap GREEN_WHITE_LINEAR
Description
A linear green and white colormap.

PRISM
public Imsl.Chart2D.Colormap PRISM

Description
A prism colormap.

RED
public Imsl.Chart2D.Colormap RED

Description
A linear red colormap.

RED_PURPLE
public Imsl.Chart2D.Colormap RED_PURPLE

Description
A red and purple colormap.

RED_TEMPERATURE
public Imsl.Chart2D.Colormap RED_TEMPERATURE

Description
A linear red temperature colormap.

SPECTRAL
public Imsl.Chart2D.Colormap SPECTRAL

Description
A spectral colormap.

STANDARD_GAMMA
public Imsl.Chart2D.Colormap STANDARD_GAMMA

Description
A standard gamma colormap.

WB_LINEAR
public Imsl.Chart2D.Colormap WB_LINEAR

Description
A linear black and white (grayscale) colormap, the reverse of BW_LINEAR.
Chapter 25: Quality Control and Improvement Charts

Types

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShewhartControlChart</td>
<td>1521</td>
</tr>
<tr>
<td>ControlLimit</td>
<td>1527</td>
</tr>
<tr>
<td>XbarR</td>
<td>1529</td>
</tr>
<tr>
<td>RChart</td>
<td>1536</td>
</tr>
<tr>
<td>XbarS</td>
<td>1539</td>
</tr>
<tr>
<td>SChart</td>
<td>1546</td>
</tr>
<tr>
<td>XmR</td>
<td>1550</td>
</tr>
<tr>
<td>NpChart</td>
<td>1553</td>
</tr>
<tr>
<td>PChart</td>
<td>1556</td>
</tr>
<tr>
<td>CChart</td>
<td>1560</td>
</tr>
<tr>
<td>UChart</td>
<td>1563</td>
</tr>
<tr>
<td>EWMA</td>
<td>1566</td>
</tr>
<tr>
<td>CuSum</td>
<td>1569</td>
</tr>
<tr>
<td>CuSumStatus</td>
<td>1572</td>
</tr>
<tr>
<td>ParetoChart</td>
<td>1578</td>
</tr>
</tbody>
</table>

ShewhartControlChart Class

public class Imsl.Chart2D.QC.ShewhartControlChart : Data

ShewhartControlChart is the base class for the Shewhart control charts.

The control limits are generally calculated as

\[ \text{center} + k\sigma \]
where the meaning of \textit{center} and $\sigma$ depend on the specific control chart being drawn. The variable $k$ is the value of the attribute “ControlLimit” associated with the line being drawn. Typically there are three lines with $k = -3, 0, 3$.

If all of the sample sizes are equal, the lines are horizontal. If the sample sizes are unequal, the lines have a “stairstep” pattern. Horizontal lines have a title drawn just above the line and right-adjusted on the chart. The contents of the title is determined by the line’s “Title” attribute. If the title contains the placeholder “\{0\}”, it is replaced by the lines value. This replacement is done using \texttt{System.String.Format}.

### Fields

**d2_data**

public double[] d2_data

\textbf{Description}

This field contains $d_{2,n}$ the mean of the ranges of $n$ samples from a Gaussian distribution. The entries for $n= 0$ and $1$ are NaN because these values are not defined. Valid entries in the table are for $n=2$ through $50$.

**d3_data**

public double[] d3_data

\textbf{Description}

This field contains $d_{3,n}$ the standard deviation of the ranges of $n$ samples from a Gaussian distribution. The entries for $n= 0$ and $1$ are NaN because these values are not defined. Valid entries in the table are for $n=2$ through $50$.

### Properties

**Center**

virtual public double Center {get; set; }

\textbf{Description}

The value of the attribute “Center”. This is used to position the center line.

**CenterLine**

virtual public Imsl.Chart2D.QC.ControlLimit CenterLine {get; }

\textbf{Description}

The center line.

\textbf{Property Value}

Its default value is zero.
ControlData
virtual public Imsl.Chart2D.Data ControlData {get; }

Description
The Data object for the control data.

LowerControlLimit
virtual public Imsl.Chart2D.QC.ControlLimit LowerControlLimit {get; }

Description
The lower control limit.

MeanSampleSize
virtual public double MeanSampleSize {get; }

Description
Value of the attribute “MeanSampleSize”, the average sample size.

Property Value
Its default value is zero.

UpperControlLimit
virtual public Imsl.Chart2D.QC.ControlLimit UpperControlLimit {get; }

Description
The upper control limit.

Constructor

ShewhartControlChart
public ShewhartControlChart(Imsl.Chart2D.AxisXY axis)

Description
Constructs a Shewhart control chart.

Parameter
axis – The AxisXY parent of this node.

Methods

AddCenterLine
virtual public Imsl.Chart2D.QC.ControlLimit AddCenterLine()
Description
Adds the center line to the control chart and returns the newly added line.

Returns
The ControlLimit object which draws the center line.

**AddControlLimit**

```csharp
virtual public Imsl.Chart2D.QC.ControlLimit AddControlLimit()
```

Description
Adds a control limit to the chart. These can be used as tolerance limit lines or warning control limit lines.

Returns
The newly created ControlLimit object.

**AddLowerControlLimit**

```csharp
virtual public Imsl.Chart2D.QC.ControlLimit AddLowerControlLimit()
```

Description
Creates lower ControlLimit, adds it to the control chart, and returns the newly created object.

Returns
The ControlLimit object which draws the lower control limit line.

**AddUpperControlLimit**

```csharp
virtual public Imsl.Chart2D.QC.ControlLimit AddUpperControlLimit()
```

Description
Creates upper ControlLimit, adds it to the control chart, and returns the newly created object.

Returns
The ControlLimit object which draws the upper control limit line.

**AddWecoLimits**

```csharp
virtual public Imsl.Chart2D.QC.ControlLimit[] AddWecoLimits()
```

Description
Adds lines for the Western Electric Company Rules.

Returns
An array containing the four added lines. They are in order with the ControlLimit corresponding to $k = -2$ first and the ControlLimit corresponding to $k = +2$ last.
Remarks
These additional lines are at \( center + k\sigma \) for \( k = -2, -1, 1, 2 \).

The meaning of \( center \) and \( \sigma \) depend on the chart to which these lines are added. The “LineColor” attribute for these lines is set to yellow.

**GetSampleSize**
virtual public int[] GetSampleSize()

**Description**
Returns the value of the attribute “SampleSize”.

**Returns**
The value of the attribute “SampleSize”. Its default value is an array of length one containing a one, \((\text{new int[]}{\{1\}})\).

**Paint**
override public void Paint(Imsl.Chart2D.Draw draw)

**Description**
Paints this node and all of its children. This is normally called only by Paint (p. 1372) in this node’s parent.

**Parameter**
- \( \text{draw} \) – A Draw object which is used for drawing the chart.

**RemoveControlLimit**
virtual public void RemoveControlLimit(Imsl.Chart2D.QC.ControlLimit line)

**Description**
Removes a control limit from the chart.

**Parameter**
- \( \text{line} \) – The ControlLimit object to be removed from this chart.

**SetData**
virtual public Imsl.Chart2D.Data SetData(int[] y)

**Description**
Sets the integer data in the control chart.

**Parameter**
- \( \text{y} \) – An array containing the data values.

**Returns**
A Data object containing the data points.
Remarks
By default, the data points are drawn as filled circle markers connected by a line.

SetData
virtual public Imsl.Chart2D.Data SetData(double[] y)

Description
Sets the data in the control chart. By default, the data points are drawn as filled circle markers connected by a line.

Parameter
  y – An array containing the data values.

Returns
A Data object containing the data points.

SetDataRange
override public void SetDataRange(double[] range)

Description
Update the data range, range = {xmin, xmax, ymin, ymax}

Parameter
  range – A double array which contains the updated range, {xmin, xmax, ymin, ymax}

Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

SetSampleSize
virtual public void SetSampleSize(int sampleSize)

Description
Sets the value of the attribute “SampleSize” when there is only a single sample size.

Parameter
  sampleSize – The value of the attribute “SampleSize”.

Remarks
Its default value is an array of length one containing a one, (new int[] {1}).

SetSampleSize
virtual public void SetSampleSize(int[] sampleSize)

Description
Sets the value of the attribute “SampleSize”. Its default value is an array of length one containing a one, (new int[] {1}).
Parameter

sampleSize – The value of the attribute “SampleSize”.

SetX

virtual public void SetX(double[] x)

Description

Sets the x-coordinates of the data. The “X” chart attribute of each ControlLimit added to this chart is also set. If not set, the values 0, 1, 2, ... are used.

Parameter

x – An array containing the x-coordinates. The length of x must equal the length of the data array used to construct this object.

ControlLimit Class

public class Imsl.Chart2D.QC.ControlLimit : Data

ControlLimit is a control limit line on a process control chart.

This class draws either a horizontal line or a stair step line depending on the value of the attribute “Value”. Its value is an array. If the array has a just one entry then a horizontal line is drawn at y equal to this value. This line extends across the limit given by the x-axis window attribute. If the array has more than one entry then a stair step line is drawn using the array values as the y-coordinates of the stair step line.

Properties

ControlLimitValue

virtual public double ControlLimitValue {get; set; }

Description

The attribute “ControlLimitValue”. This is the y-coordinate at which the line is drawn. Its default value is zero.

MaximumValue

virtual public double MaximumValue {get; set; }

Description

The maximum value of this control limit line.
Property Value
Its default value is positive infinity.

MinimumValue
virtual public double MinimumValue {get; set; }

Description
The minimum value of this control limit line.

Property Value
Its default value is negative infinity.

Methods

GetValue
virtual public double[] GetValue()

Description
Returns the value of this control limit line.

Returns
The y-coordinate at which this control limit line is drawn.

Paint
override public void Paint(Imsl.Chart2D.Draw draw)

Description
Paints the horizontal control limit line as wide as the window.

Parameter
   draw – A Draw object which is used for painting.

Remarks
This is normally called only by the Paint method in this node’s parent.

SetDataRange
override public void SetDataRange(double[] range)

Description
Update the data range, range = {xmin,xmax,ymin,ymax}

Parameter
   range – A double array which contains the updated range, {xmin,xmax,ymin,ymax}
Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

SetValue
virtual public void SetValue(double y)

Description
Sets the value of this control limit line. The actual value used is subject to minimum and maximum values set as attributes to this object.

Parameter

y – The y-coordinate at which this control limit line is drawn.

SetValue
virtual public void SetValue(double[] y)

Description
Sets the value of this control limit line to an array of values.

Parameter

y – An array containing the y-coordinates of a stair step line.

XbarR Class

public class Imsl.Chart2D.QC.XbarR : ShewhartControlChart

XbarR is an X-bar chart for monitoring a process using sample ranges.
The control limits are at

$$\bar{x} + k \frac{\bar{R}}{d_{2n} \sqrt{n}}$$

where $\bar{x}$ is the grand mean of all of the observations, $\bar{R}$ is the mean of the observed ranges, $n$ is the sample size, and $k$ is the value of the “ControlLimit” attribute for the limit. Additionally, $d_{2n} = E[R]/\sigma$, where $R$ is the range of data from a Gaussian distribution. Therefore $\bar{R}/d_{2n}$ is an estimate of the within sample standard deviation.

By default, the chart contains an upper control limit with $k=3$, a lower control limit with $k=-3$, and a central line equal to $\bar{x}$. Additional control limits can be added. The method AddWeco adds control limits with $k = -2, -1, 1, 2$. 
### Property

**Rbar**

```csharp
virtual public double Rbar {get; set; }
```

**Description**
The value of the “Rbar” attribute, the mean of the ranges for a series of samples.

### Constructors

**XbarR**

```csharp
public XbarR(Imsl.Chart2D.AxisXY axis, double[,] x)
```

**Description**
Creates an X-bar chart from sample data using sample ranges.

**Parameters**
- **axis** – The AxisXY parent of this node.
- **x** – An array of arrays containing sample data. The data of the i-th sample is in x[i]. Each row must have between 2 and 50.

**XbarR**

```csharp
public XbarR(Imsl.Chart2D.AxisXY axis, double[,] x)
```

**Description**
Creates an X-bar chart from sample data using sample ranges.

**Parameters**
- **axis** – The AxisXY parent of this node.
- **x** – A two-dimensional array containing sample data. The data of the i-th sample is in x[i,*]. The number of columns must be between 2 and 50 entries.

**XbarR**

```csharp
public XbarR(Imsl.Chart2D.AxisXY axis, int sampleSize, double[] xbar, double[] range)
```

**Description**
Creates an X-bar control chart given the means and ranges for a series of equally sized samples.

**Parameters**
- **axis** – The AxisXY parent of this node.
- **sampleSize** – The number of observations in each sample. It must be between 2 and 50.
- **xbar** – An array containing the mean values for a series of samples.
range – An array containing the ranges of the samples. The arrays xbar and range must have the same lengths.

**XbarR**

public XbarR(Imsl.Chart2D.AxisXY axis, int[] sampleSize, double[] xbar, double[] range)

**Description**

Creates an X-bar control chart given the means and ranges for a series of unequally sized samples.

**Parameters**

- axis – The AxisXY parent of this node.
- sampleSize – An array containing the number of observations in each sample. Each sample must contain between 2 and 50 observations.
- xbar – An array containing the mean values for a series of samples.
- range – An array containing the ranges of the samples.

**Methods**

**CapabilityIndexCp**

virtual public double CapabilityIndexCp(double lowerSpecificationLimit, double upperSpecificationLimit)

**Description**

Returns the capability index $c_p$.

**Parameters**


**Returns**

The capability index $c_p$.

**Remarks**

The capability index is

$$c_p = \frac{\text{USL} - \text{LSL}}{6\sigma}$$

where LSL and USL are the lower and upper specification limits and

$$\sigma = \frac{\bar{R}}{d_{2,n}\sqrt{n}}$$

**CapabilityIndexCpk**

virtual public double CapabilityIndexCpk(double lowerSpecificationLimit, double upperSpecificationLimit)
Description
Returns the capability index \( c_{pk} \).

Parameters

Returns
The capability index \( c_{pk} \).

Remarks
The capability index is
\[
c_{pk} = \min \left\{ \frac{USL - \bar{x}}{3\sigma}, \frac{\bar{x} - LSL}{3\sigma} \right\}
\]
where LSL and USL are the lower and upper specification limits, \( \bar{x} \) is the center line, and
\[
\sigma = \frac{\bar{R}}{d_{2,n}\sqrt{n}}
\]

CreateCharts
static public Imsl.Chart2D.QC.ShewhartControlChart[] CreateCharts(Imsl.Chart2D.Chart chart, double[][] x)

Description
Creates a combined XbarR chart and RChart from data.

Parameters
- chart – The Chart object which is the parent of the two charts being created.
- x – An array of arrays containing sample data. The data of the \( i \)-th sample is in \( x[i] \). Each sample must contain at least 2 and no more than 50 observations.

Returns
An array of length two containing the XbarR chart and the RChart.

Remarks
The viewport of the XbarR chart is [0.2, 0.9] by [0.1, 0.4]. The viewport of the RChart chart is [0.2, 0.9] by [0.5, 0.8].
Parameters

- chart – The Chart object which is the parent of the two charts being created.
- x – A two-dimensional array containing sample data. The data of the i-th sample is in x[i, *].
  Each sample must contain at least 2 and no more than 50 observations.

Returns

An array of length two containing the XbarR chart and the RChart.

Remarks

The viewport of the XbarR chart is [0.2, 0.9] by [0.1, 0.4]. The viewport of the RChart chart is [0.2, 0.9]
by [0.5, 0.8].

CreateCharts

static public Imsl.Chart2D.QC.ShewhartControlChart[]
CreateCharts(Imsl.Chart2D.Chart chart, int sampleSize, double[] xbar, double[] range)

Description

Creates a combined XbarR chart and RChart given the means and ranges for a series of equally sized samples.

Parameters

- chart – The Chart object which is the parent of the two charts being created.
- sampleSize – The number of observations in each sample. It must be between 2 and 50.
- xbar – An array containing the mean values for a series of samples.
- range – An array containing the ranges of the samples.

Returns

An array of length two containing the XbarR chart and the RChart.

Remarks

The viewport of the XbarR chart is [0.2, 0.9] by [0.1, 0.4]. The viewport of the RChart chart is [0.2, 0.9]
by [0.5, 0.8].

CreateCharts

static public Imsl.Chart2D.QC.ShewhartControlChart[]
CreateCharts(Imsl.Chart2D.Chart chart, int[] sampleSize, double[] xbar, double[] range)

Description

Creates a combined XbarR chart and RChart given the means and ranges for a series of unequally sized samples.
Parameters

chart – The Chart object which is the parent of the two charts being created.
sampleSize – An array containing the number of observations in each sample. Each sample must have between 2 and 50 observations.
xbar – An array containing the mean values for a series of samples.
range – An array containing the ranges of the samples.

Returns

An array of length two containing the XbarR chart and the RChart.

Remarks

The viewport of the XbarR chart is [0.2, 0.9] by [0.1, 0.4]. The viewport of the RChart chart is [0.2, 0.9] by [0.5, 0.8].

PrePaint

override public void PrePaint()

Description

Setup chart with current settings.

Example: XbarR Chart

During a manufacturing process 15 samples, each containing 5 items, were measured. An XbarR chart was contructed from the 15 sample ranges.

using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class XbarREx1 : FrameChart
{
    static private readonly double[][] data = {
        new double[]{44.73, 45.47, 45.39, 45.33, 45.24},
        new double[]{45.57, 46.87, 45.40, 46.68, 44.29},
        new double[]{46.39, 45.31, 46.74, 46.06, 45.51},
        new double[]{46.54, 46.27, 44.57, 45.36, 45.72},
        new double[]{45.58, 45.59, 46.02, 45.45, 46.42},
        new double[]{46.91, 45.38, 49.48, 44.91, 45.17},
        new double[]{45.98, 45.29, 45.50, 45.77, 46.44},
        new double[]{46.30, 45.65, 45.21, 45.43, 45.57},
        new double[]{45.77, 45.38, 45.65, 45.25, 45.89},
        new double[]{44.10, 45.44, 45.27, 45.53, 44.65},
        new double[]{45.95, 46.22, 46.71, 45.92, 45.90},
        new double[]{44.87, 44.98, 45.91, 45.18, 45.64},
        new double[]{44.70, 45.89, 46.67, 45.84, 45.07},
        new double[]{45.90, 45.30, 46.30, 46.34, 46.34},
        new double[]{44.90, 46.23, 45.31, 45.29, 45.16}
    };

    public XbarREx1()
Output
public class Imsl.Chart2D.QC.RChart : ShewhartControlChart

RChart is an R chart using sample ranges to monitor the variability of a process. Each sample must contain at least two observations. The range of a sample is the maximum observed value minus the minimum observed value.

The control limits are at

\[ \bar{R} + k \frac{d_{3,n} \bar{R}}{d_{2,n}} \]

\( \bar{R} \) is the mean of the observed ranges, \( n \) is the sample size, and \( k \) is the value of the “ControlLimit” attribute for the line. Additionally, \( d_{2,n} \) is the mean of the distribution of the ranges of \( n \) samples from the normal distribution with mean zero and standard deviation one. The standard deviation of this distribution is \( d_{3,n} \). Therefore

\[ \frac{d_{3,n} \bar{R}}{d_{2,n}} \]

is an estimator of the standard deviation of the ranges.

By default, the chart contains an upper control limit line with \( k=3 \), a lower control limit line with \( k=-3 \), and a central line with \( k=0 \). Additional control limit lines can be added. The method AddWeco adds control limit lines with \( k = -2, -1, 1, 2 \).

Constructors

RChart

public RChart(Imsl.Chart2D.AxisXY axis, double[,] x)

Description

Creates an R chart given sample data.

Parameters

axis – The AxisXY parent of this node.

x – An array of arrays containing sample data. The data of the \( i \)-th sample is in \( x[i] \). Each sample must contain at least 2 and no more than 50 observations.

Exception

System.ArgumentException is thrown if the number of samples is less than 2 or greater than 50.

RChart

public RChart(Imsl.Chart2D.AxisXY axis, double[,] x)
Description
Creates an R chart given sample data.

Parameters
- axis – The AxisXY parent of this node.
- x – A two-dimensional array containing sample data. The data of the i-th sample is in x[i,*]. Each sample must contain at least 2 and no more than 50 observations.

Exception
System.ArgumentException is thrown if the number of samples is less than 2 or greater than 50.

RChart
public RChart(Imsl.Chart2D.AxisXY axis, int sampleSize, double[] range)

Description
Creates an R chart given the ranges for a series of equally sized samples.

Parameters
- axis – The AxisXY parent of this node.
- sampleSize – The number of observations in each sample. It must be at least 2 and no more than 50.
- range – An array containing the data ranges for a series of samples.

RChart
public RChart(Imsl.Chart2D.AxisXY axis, int[] sampleSize, double[] range)

Description
Creates an R chart given the means for a series of equally sized samples.

Parameters
- axis – The AxisXY parent of this node.
- sampleSize – An array containing the number of observations in each sample. It must be at least 2 and no more than 50.
- range – An array containing the data ranges for a series of samples.

Method
PrePaint
override public void PrePaint()

Description
Setup chart with current settings.
Example: R Chart

During a manufacturing process 15 samples, each containing 3 to 5 items, were measured. An R chart was constructed from the 15 sample ranges. Since the sample sizes are unequal the upper control limit is a stair step line.

```csharp
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class RChartEx1 : FrameChart
{
    static private double[][] data = {
        new double[]{44.73, 45.47, 45.39, 45.33, 45.24},
        new double[]{45.57, 46.87, 45.40},
        new double[]{46.39, 45.31, 46.74, 46.06, 45.51},
        new double[]{45.54, 46.27, 44.57, 45.36},
        new double[]{45.58, 45.59, 46.02, 45.45, 46.42},
        new double[]{45.91, 45.38, 44.98, 44.91, 45.17},
        new double[]{45.98, 45.29, 45.50, 45.77, 46.44},
        new double[]{46.30, 45.65, 45.21, 45.43},
        new double[]{45.77, 45.38, 45.65, 45.25, 45.89},
        new double[]{44.10, 46.35, 44.65},
        new double[]{45.95, 46.22, 45.92, 45.90},
        new double[]{44.87, 44.98, 45.91, 45.18, 45.64},
        new double[]{44.70, 45.89, 45.84, 45.07},
        new double[]{45.90, 45.30, 46.30, 46.34, 46.34},
        new double[]{44.90, 46.23, 45.16}
    };

    public RChartEx1()
    {
        AxisXY axis = new AxisXY(this.Chart);
        new RChart(axis, data);
    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new RChartEx1());
    }
}
```
XbarS Class

public class Imsl.Chart2D.QC.XbarS : ShewhartControlChart

XbarS is an X-bar chart for monitoring a process using sample standard deviations. The control limits are at

$$\bar{x} + k \frac{\bar{s}}{c_{4,n} \sqrt{n}}$$

Output

![Graph of XbarS Chart](image-url)
where $\bar{x}$ is the grand mean of all of the observations, $n$ is the sample size, and $k$ is the value of the “ControlLimit” attribute for the limit. Additionally, $c_{4,n}$ is a factor such that $\bar{s}/c_{4,n}$ is an unbiased estimator of the within sample standard deviation.

By default, the chart contains an upper control limit line with $k=3$, a lower control limit line with $k=-3$, and a central line equal to $\bar{x}$. Additional control limit lines can be added. The method AddWeco adds control limits with $k = -2, -1, 1, 2$.

**Property**

**Wbar**

```csharp
virtual public double Wbar {get; set; }
```

**Description**

The value of the “Wbar” attribute.

**Remarks**

This is the within sample variation for a series of samples. Its default value is set by the constructor to an estimate of the within sample variation.

**Constructors**

**XbarS**

```csharp
public XbarS(Imsl.Chart2D.AxisXY axis, double[,] x)
```

**Description**

Creates an XBarS chart from sample data using within sample standard deviations.

**Parameters**

- `axis` – The AxisXY parent of this node.
- `x` – An array of arrays containing sample data. The data of the $i$-th sample is in $x[i]$. Each row must have at least one entry.

**XbarS**

```csharp
public XbarS(Imsl.Chart2D.AxisXY axis, double[,] x)
```

**Description**

Creates an XBarS chart from sample data using within sample standard deviations.

**Parameters**

- `axis` – The AxisXY parent of this node.
- `x` – A two-dimensional array containing sample data. The data of the $i$-th sample is in $x[i,*]$. 

1540 • XbarS

**IMSL C# Numerical Library**
XbarS
public XbarS(Imsl.Chart2D.AxisXY axis, int sampleSize, double[] xbar, double[] w)

Description
Creates an XBarS chart given the means and standard deviations for a series of equally sized samples.

Parameters
axis – The AxisXY parent of this node.
sampleSize – The number of observations in each sample. It must be at least one.
xbar – An array containing the mean values for a series of samples.
w – An array containing the within sample variation for a series of samples.

Exception
System.ArgumentException is thrown if the two input arrays do not have the same length.

XbarS
public XbarS(Imsl.Chart2D.AxisXY axis, int[] sampleSize, double[] xbar, double[] w)

Description
Creates an XBarS chart given the means and standard deviations for a series of unequally sized samples.

Parameters
axis – The AxisXY parent of this node.
sampleSize – An array containing the number of observations in each sample. Each sample must have at least one observation.
xbar – An array containing the mean values for a series of samples.
w – An array containing the within sample variation for a series of samples.

Exception
System.ArgumentException is thrown if the three input arrays do not all have the same length.

Methods
CapabilityIndexCp
virtual public double CapabilityIndexCp(double lowerSpecificationLimit, double upperSpecificationLimit)

Description
Returns the capability index $c_p$.

Quality Control and Improvement Charts
Parameters


Returns
The capability index $c_p$.

Remarks
The capability index is

$$c_p = \frac{USL - LSL}{6\sigma}$$

where LSL and USL are the lower and upper specification limits and

$$\sigma = \frac{\bar{x}}{c_{4,n}\sqrt{n}}$$

CapabilityIndexCpk
virtual public double CapabilityIndexCpk(double lowerSpecificationLimit, double upperSpecificationLimit)

Description
Returns the capability index $c_{pk}$.

Parameters


Returns
The capability index $c_{pk}$.

Remarks
The capability index is

$$c_{pk} = \min \left( \frac{USL - \bar{x}}{3\sigma}, \frac{\bar{x} - LSL}{3\sigma} \right)$$

where LSL and USL are the lower and upper specification limits, $\bar{x}$ is the center line, and

$$\sigma = \frac{\bar{x}}{c_{4,n}\sqrt{n}}$$

CreateCharts
static public Imsl.Chart2D.QC.ShewhartControlChart[] CreateCharts(Imsl.Chart2D.Chart chart, double[][] x)

Description
Creates a combined XBarS chart and SChart from data.
Parameters

chart – The Chart object which is the parent of the two charts being created.

x – An array of arrays containing sample data. The data of the i-th sample is in x[i]. Each row must have at least one entry.

Returns

An array of length two containing the XBarS chart and the SChart.

Remarks

The viewport of the XBarS chart is [0.2, 0.9] by [0.1, 0.4]. The viewport of the SChart chart is [0.2, 0.9] by [0.5, 0.8].

CreateCharts

static public Imsl.Chart2D.QC.ShewhartControlChart[] CreateCharts(Imsl.Chart2D.Chart chart, double[,] x)

Description

Creates a combined XBarS chart and SChart from data.

Parameters

chart – The Chart object which is the parent of the two charts being created.

x – A two-dimensional array containing sample data. The data of the i-th sample is in x[i,*].

Returns

An array of length two containing the XBarS chart and the SChart.

Remarks

The viewport of the XBarS chart is [0.2, 0.9] by [0.1, 0.4]. The viewport of the SChart chart is [0.2, 0.9] by [0.5, 0.8].

CreateCharts

static public Imsl.Chart2D.QC.ShewhartControlChart[] CreateCharts(Imsl.Chart2D.Chart chart, int sampleSize, double[] xbar, double[] w)

Description

Creates a combined XBarS chart and SChart given the means and in sample standard deviations for a series of equally sized samples.

Parameters

chart – The Chart object which is the parent of the two charts being created.

sampleSize – The number of observations in each sample. It must be at least one.

xbar – An array containing the mean values for a series of samples.

w – An array containing the in sample standard deviations of the samples.

Returns

An array of length two containing the XBarS chart and the SChart.
Remarks
The viewport of the XBarS chart is [0.2, 0.9] by [0.1, 0.4]. The viewport of the SChart chart is [0.2, 0.9] by [0.5, 0.8].

CreateCharts
static public Imsl.Chart2D.QC.ShewhartControlChart[] CreateCharts(Imsl.Chart2D.Chart chart, int[] sampleSize, double[] xbar, double[] w)

Description
Creates a combined X-bar chart and S-chart given the means and in sample standard deviations for a series of unequally sized samples.

Parameters
- chart – The Chart object which is the parent of the two charts being created.
- sampleSize – An array containing the number of observations in each sample. Each sample must have at least one observation.
- xbar – An array containing the mean values for a series of samples.
- w – An array containing the in sample standard deviations of the samples.

Returns
An array of length two containing the XBarS chart and the SChart.

PrePaint
override public void PrePaint()

Description
Setup chart with current settings.

Example: XbarS Chart
During a manufacturing process 15 samples, each containing 12 items, were measured. An XbarS chart was constructed from the 15 sample standard deviations.

using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class XbarSEx1 : FrameChart
{
    static private readonly double[][] data = {
        new double[] {23.97, 24.08, 23.16, 23.49, 24.73, 25.26, 22.97, 23.12, 24.66, 24.20, 24.62, 24.56},
        new double[] {24.20, 24.50, 23.45, 22.22, 25.10, 24.41, 24.05, 23.89, 24.83, 25.21, 23.70},
        new double[] {23.73, 22.70, 23.54, 24.37, 24.08, 23.74, 24.08, 23.95, 24.20, 23.43, 24.26, 23.61},
        new double[] {23.46, 23.14, 23.96, 23.37, 23.73, 24.29, 24.13, 23.62, 24.08},

        // More sample data...
    }
new double[][]{23.95, 24.13, 22.95, 24.72, 24.40, 22.82, 22.66, 22.71, 24.21, 
23.39, 23.41, 22.56},
new double[][]{24.13, 24.28, 23.84, 24.55, 23.53, 23.77, 24.38, 22.58, 24.47, 
23.63, 22.64, 24.12},
new double[][]{23.69, 24.19, 24.76, 23.29, 24.84, 24.12, 23.83, 22.60, 24.35, 
22.96, 23.81, 23.46},
new double[][]{24.35, 23.11, 25.24, 24.10, 24.93, 22.93, 23.47, 23.55, 23.91, 
24.08, 22.45, 24.13},
new double[][]{24.98, 24.58, 23.52, 24.42, 23.90, 23.55, 23.67, 24.25, 23.85, 
23.08, 23.44, 23.43},
new double[][]{23.90, 24.04, 24.29, 23.62, 23.29, 23.16, 24.34, 24.37, 24.19, 
24.33, 22.17, 23.66},
new double[][]{23.51, 24.98, 24.34, 23.87, 23.29, 23.96, 23.06, 23.47, 23.53, 
22.87, 23.38, 22.86},
new double[][]{23.13, 23.17, 23.40, 23.68, 23.41, 23.67, 23.37, 24.40, 24.64, 
24.16, 24.17, 23.88},
new double[][]{23.52, 24.23, 24.25, 24.31, 23.89, 24.02, 24.04, 23.83, 22.82, 
23.93, 23.55, 23.40},
new double[][]{23.66, 23.59, 23.79, 24.07, 23.76, 23.34, 23.65, 23.73, 25.12, 
23.65, 23.23, 23.13},
new double[][]{23.04, 23.75, 22.84, 23.46, 21.72, 23.81, 24.51, 24.01, 24.73, 
23.88, 23.34, 24.98}
};

public XbarSE1()
{
    XbarS.CreateCharts(this.Chart, data);
}

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new XbarSEx1());
}


**SChart Class**

```csharp
public class IMsl.Chart2D.QC.SChart : ShewhartControlChart

SChart is an S chart using sample standard deviations to monitor the variability of a process. This is normally used with sample sizes greater than 10.
```
The control limits are at
\[ \bar{s} + k \frac{\bar{s}}{c_{4,n}} \sqrt{1 - \frac{c_{4,n}^2}{n}} \]
where \( \bar{s} \) is the mean of the within sample standard deviations, \( n \) is the sample size, and \( k \) is the value of the “ControlLimit” attribute for the line. Additionally,
\[ c_{4,n} = \sqrt{\frac{2}{n-1} \frac{\Gamma\left(\frac{n}{2}\right)}{\Gamma\left(\frac{n+1}{2}\right)}} \]
is a factor such that \( \bar{s}/c_{4,n} \) is an unbiased estimator of the within sample standard deviation. By default, the chart contains an upper control limit line with \( k=3 \), a lower control limit line with \( k=-3 \), and a central line equal to \( \bar{s} \). Additional control limit lines can be added. The method \( \text{AddWeco} \) adds control limits with \( k = -2, -1, 1, 2 \).

**Constructors**

**SChart**

```java
public SChart(Imsl.Chart2D.AxisXY axis, double[][] x)
```

**Description**

Creates an S chart given sample data.

**Parameters**

- `axis` – The `AxisXY` parent of this node.
- `x` – An array of arrays containing sample data. The data of the \( i \)-th sample is in \( x[i] \). Each row must have at least two entries.

**SChart**

```java
public SChart(Imsl.Chart2D.AxisXY axis, double[,] x)
```

**Description**

Creates an S chart given sample data.

**Parameters**

- `axis` – The `AxisXY` parent of this node.
- `x` – A two-dimensional array containing sample data. The data of the \( i \)-th sample is in \( x[i,*] \). There must be at least two columns.

**SChart**

```java
public SChart(Imsl.Chart2D.AxisXY axis, int sampleSize, double[] s)
```

**Description**

Creates an S chart given the within sample standard deviations for a series of equally sized samples.
Parameters
- axis – The AxisXY parent of this node.
- sampleSize – The number of observations in each sample. It must be at least 2.
- s – An array containing the within sample standard deviations for a series of samples.

SChart
public SChart(Imsl.Chart2D.AxisXY axis, int[] sampleSize, double[] s)

Description
Creates an S chart given the within sample standard deviations for a series of unequally sized samples.

Parameters
- axis – The AxisXY parent of this node.
- sampleSize – An array containing the number of observations in each sample. All samples must have at least two observations.
- s – An array containing the within sample standard deviations for a series of samples.

Exception
System.ArgumentException is thrown if the two input arrays do not have the same length.

Method
PrePaint
override public void PrePaint()

Description
Setup chart with current settings.

Example: S Chart
During a manufacturing process 15 samples, each containing up to 12 items, were measured. An S chart was constructed from the 15 sample standard deviations. Since the sample sizes are unequal the upper and lower control limit lines are stair step lines.

```csharp
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class SChartEx1 : FrameChart
{
    static private double[][] data = {
        new double[]{23.97, 24.08, 23.16, 23.49, 24.73, 25.26, 22.97, 23.12, 24.66,
            24.20, 24.62, 24.56},
        new double[]{24.20, 24.50, 23.45, 24.05, 23.75, 23.89, 24.83, 25.21, 23.70},
        new double[]{23.73, 22.70, 23.54, 24.37, 23.74, 24.08, 23.95, 24.20, 23.43,
```
new double[]{24.26, 23.61},
new double[]{23.46, 23.14, 23.73, 24.29, 24.13, 23.62, 24.08, 23.73, 23.91, 23.65},
new double[]{23.95, 22.95, 24.72, 24.40, 22.82, 22.66, 22.71, 23.41, 22.56},
new double[]{24.13, 24.28, 23.84, 24.55, 22.58, 24.47, 23.63, 22.64, 24.12},
new double[]{23.69, 24.19, 24.76, 23.83, 22.60, 24.35, 22.96, 23.81, 23.46},
new double[]{24.98, 24.58, 23.52, 23.55, 23.67, 24.25, 23.85, 23.08, 23.44, 23.43},
new double[]{23.51, 23.87, 23.29, 23.96, 23.06, 23.47, 23.53, 22.87, 23.38, 22.86},
new double[]{23.13, 23.17, 23.40, 23.68, 23.37, 24.40, 24.64, 24.16, 24.17, 23.88},
new double[]{23.52, 24.23, 24.25, 23.83, 22.82, 23.93, 23.55, 23.40},
new double[]{23.66, 23.59, 23.79, 24.07, 23.65, 23.73, 25.12, 23.65, 23.23, 23.13},
new double[]{23.04, 23.75, 22.84, 23.46, 21.72, 24.73, 23.88, 23.34, 24.98}];

public SChartEx1()
{
    AxisXY axis = new AxisXY(this.Chart);
    new SChart(axis, data);
}

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new SChartEx1());
}
XmR Class

public class Imsl.Chart2D.QC.XmR : ShewhartControlChart

XmR is an XmR chart for monitoring a process using moving ranges.

The moving range control chart uses the moving range of two successive observations to measure the process variability. This control chart is used for individual measurements (sample size = 1).
The moving range is defined to be \( MR_i = |x_i - x_{i-1}| \).

The control limits are at

\[
\bar{x} + k \frac{MR}{d_{2,2}}
\]

where \( \bar{x} \) is the mean of all of the individual observations, \( \overline{MR} \) is the mean of the moving averages, and \( k \) is the value of the “ControlLimit” attribute for the line. Additionally, \( d_{2,n} = \frac{E(R)}{\sigma} \) where \( R \) is the range of a Gaussian distribution. Therefore \( \overline{MR}/d_{2,2} \) is an estimate of the standard deviation.

By default, the chart contains an upper control limit line with \( k=3 \), a lower control limit line with \( k=-3 \), and a central line equal to \( \bar{x} \). Additionally control limits can be added. The method `AddWeco` adds control limits with \( k = -2, -1, 1, 2 \).

### Property

**MRBar**

virtual public double MRBar {get; set; }

**Description**

The expected mean of all of the moving ranges of two observations.

**Property Value**

Its default value is the computed from the data passed to the constructor.

### Constructor

**XmR**

public XmR(Imsl.Chart2D.AxisXY axis, double[] x)

**Description**

Creates an XmR chart given sample data.

**Parameters**

- `axis` – The AxisXY parent of this node.
- `x` – An array containing sample data.

### Method

**PrePaint**

override public void PrePaint()
Description
Setup chart with current settings.

Example: Moving Range Chart
This moving range chart plots the flowrate for 10 batches. The data is from NIST Engineering Statistics Handbook: Individuals Control Charts.

using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class XmREx1 : FrameChart
{
    static private readonly double[] flowrate = {
        49.6, 47.6, 49.9, 51.3, 47.8, 51.2, 52.6, 52.4, 53.6, 52.1
    };

    public XmREx1()
    {
        AxisXY axis = new AxisXY(this.Chart);
        XmR mr = new XmR(axis, flowrate);
    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new XmREx1());
    }
}
NpChart Class

public class Imsl.Chart2D.QC.NpChart : ShewhartControlChart

NpChart is an np-chart for monitoring the number of defects when defects are not rare.

Control limits are computed using the binomial distribution. If defects are rare CChart (p. 1560) should be used instead.
The control limits are at

$$np + k\sqrt{np(1-p)}$$

where $p$ is the proportion of defective items, $n$ is the sample size, and $k$ is the value of the “ControlLimit” attribute for the line. By default, the chart contains an upper control limit line with $k=3$, a lower control limit line with $k=-3$, and a central line equal to $\bar{c}$. Additional control limits can be added. The method `AddWeco` adds control limits with $k = -2, -1, 1, 2$.

The lower control limit is forced to have a minimum value of zero. The y-axis labels are formatted as integers.

**Constructors**

**NpChart**

```csharp
public NpChart(Imsl.Chart2D.AxisXY axis, int sampleSize, int[] numberDefective)
```

**Description**

Creates an np-chart given the number of defects in a series of samples.

**Parameters**

- `axis` – The `AxisXY` parent of this node.
- `sampleSize` – The number of observations in each sample. It must be at least one.
- `numberDefective` – An array containing the number of defects in each of a series of samples. All of its entries must be nonnegative.

**NpChart**

```csharp
public NpChart(Imsl.Chart2D.AxisXY axis, int[] sampleSize, int[] numberDefective)
```

**Description**

Creates a np-chart given the number of defects in a series of samples, where the number of observations per sample is not constant.

**Parameters**

- `axis` – The `AxisXY` parent of this node.
- `sampleSize` – An array containing the number of observations in each sample. Each sample must contain at least one observation.
- `numberDefective` – An array containing the number of defects in each of a series of samples. All of its entries must be nonnegative.

**Method**

**PrePaint**

```csharp
override public void PrePaint()
```
**Description**

Setup chart with current settings.

**Example: np-chart**

The location of 50 chips on each of 30 successive wafers is measured. The number of defects is the number of horizontal or vertical misregistrations of chips. The data is from *NIST Engineering Statistics Handbook: Proportions Control Charts*.

```csharp
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class NpChartEx1 : FrameChart
{
    private static readonly int[] numberDefective = {
        24, 30, 16, 20, 8, 14, 32, 18, 28, 20, 10, 12,
        34, 24, 44, 16, 20, 10, 26, 22, 40, 36, 48, 30,
        18, 24, 14, 26, 18, 12
    };
    private const int sampleSize = 100;

    public NpChartEx1()
    {
        AxisXY axis = new AxisXY(this.Chart);
        axis.AxisY.AxisTitle.SetTitle("Number Defective");
        new NpChart(axis, sampleSize, numberDefective);
    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new NpChartEx1());
    }
}
```
Output

PChart Class

```csharp
public class Imsl.Chart2D.QC.PChart : ShewhartControlChart

PChart is a p-chart for monitoring the defect rate when defects are not rare.

The defect rate is the number of defects found divided by the number of samples inspected. The number of defects are not assumed to be rare. Control limits are computed using the binomial distribution. If defects are rare, use UChart (p. 1563) instead.
```
The control limits are at
\[ \hat{p} + k \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}} \]
where \( \hat{p} \) is the mean defect rate, \( n \) is the sample size, and \( k \) is the value of the “ControlLimit” attribute for the line. By default, the chart contains an upper control limit line with \( k=3 \), a lower control limit line with \( k=-3 \), and a central line equal to \( \hat{p} \). Additional control limits can be added. The method AddWeco adds control limits with \( k = -2, -1, 1, 2 \).

The true fraction conforming \( p \) can be used by setting the attribute “Center” to \( p \).

The lower control limit is forced to have a minimum value of zero. The upper control limit is forced to have a maximum value of one.

### Constructors

**PChart**

**public PChart(Imsl.Chart2D.AxisXY axis, int sampleSize, double[] defectRate)**

**Description**

Creates a p-Chart given the defect rates for a series of samples with equal sample sizes.

**Parameters**

- **axis** – The AxisXY parent of this node.
- **sampleSize** – The number of observations in each sample. It must be at least one.
- **defectRate** – An array containing defect rates of the samples. The defect rates must all be in the range [0,1].

**PChart**

**public PChart(Imsl.Chart2D.AxisXY axis, int[] sampleSize, double[] defectRate)**

**Description**

Creates a p-Chart given the defect rates for a series of samples with varying sample sizes.

**Parameters**

- **axis** – The AxisXY parent of this node.
- **sampleSize** – An array containing the number of observations in each sample. It must be at least one.
- **defectRate** – An array containing defect rates of the samples. The defect rates must all be in the range [0,1]. The lengths of the arrays sampleSize and defectRate must be equal.

**PChart**

**public PChart(Imsl.Chart2D.AxisXY axis, int sampleSize, int[] numberDefects)**

**Description**

Creates a p-Chart given the number of defects for a series of samples with equal sample sizes.
Parameters
axis – The AxisXY parent of this node.
sampleSize – The number of observations in each sample. Each sample must contain at least one observation.
numberDefects – An array containing the number of defects in each of a series of samples. The number of defects should not be less than zero.

PChart
public PChart(Imsl.Chart2D.AxisXY axis, int[] sampleSize, int[] numberDefects)

Description
Creates a p-Chart given the number of defects for a series of samples with varying sample sizes.

Parameters
axis – The AxisXY parent of this node.
sampleSize – An array containing the number of observations in each sample. It must be at least one.
numberDefects – An array of arrays containing the number of defects in each of a series of samples. The number of defects should not be less than zero.

Method
PrePaint
override public void PrePaint()

Description
Setup chart with current settings.

Example: p-Chart
The location of 50 chips on each of 30 successive wafers is measured. The number of defects is the number of horizontal or vertical misregistrations of chips. The data is from NIST Engineering Statistics Handbook: Proportions Control Charts.

```csharp
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class PChartEx1 : FrameChart
{
    static private readonly double[] defectRate = {
        0.24, 0.3, 0.16, 0.2, 0.08, 0.14, 0.32, 0.18, 0.28, 0.2,
        0.1, 0.12, 0.34, 0.24, 0.44, 0.16, 0.2, 0.1, 0.26, 0.22,
        0.4, 0.36, 0.48, 0.3, 0.18, 0.24, 0.14, 0.26, 0.18, 0.12
    };
}
```
static private readonly int sampleSize = 50;

public PChartEx1()
{
    AxisXY axis = new AxisXY(this.Chart);
    axis.AxisY.AxisTitle.SetTitle("Defect Rate");
    axis.AxisY.AxisLabel.TextFormat = "P";
    new PChart(axis, sampleSize, defectRate);
}

global::static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new PChartEx1());
}

CChart Class

public class Imsl.Chart2D.QC.CChart : ShewhartControlChart

CChart is a $c$-chart for monitoring the count of the number of defects when defects are rare. Control limits are computed using the Poisson distribution. If defects are not rare $NpChart$ (p. 1553) should be used instead.
The control limits are at

\[ \bar{c} + k \sqrt{i} \]

where \( \bar{c} \) is the mean number of defects per sample and \( k \) is the value of the “ControlLimit” attribute for the line.

By default, the chart contains an upper control limit line with \( k=3 \), a lower control limit line with \( k=-3 \), and a central line equal to \( \bar{c} \). Additional control limits can be added. The method \texttt{AddWeco} adds control limits with \( k = -2, -1, 1, 2 \).

The lower control limit is computed using the Poisson distribution. First the probability of the number of defects being less than \( k \) (the “ControlLimit” value) standard deviations from the mean in a normal distribution is computed. The number of defects required to produce the same probability assuming a Poisson distribution, with the same mean, is then computed. The lower limit is set to be one more than this number of defects.

Constructor

\textbf{CChart}

\begin{verbatim}
public CChart(Imsl.Chart2D.AxisXY axis, int[] numberDefects)

Description
Creates a C-chart given the number of defects in a series of samples.

Parameters
axis – The AxisXY parent of this node
numberDefects – The number of defects in a series of samples. The number of defects should not be less than zero. There should be the same number of items in each sample.
\end{verbatim}

Method

\textbf{PrePaint}

\begin{verbatim}
override public void PrePaint()

Description
Setup chart with current settings.
\end{verbatim}

Example: c-chart

The number of defects on each of 25 successive wafers is plotted. The data is from \textit{NIST Engineering Statistics Handbook: Counts Control Charts}.

\begin{verbatim}
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
\end{verbatim}
using System;

public class CChartEx1 : FrameChart
{
    static private readonly int[] numberOfDefects = {
        16, 14, 28, 16, 12, 20, 10, 12, 10, 17, 19, 17, 14, 16, 15, 13,
        14, 16, 11, 20, 11, 19, 16, 31, 13
    };

    public CChartEx1()
    {
        AxisXY axis = new AxisXY(this.Chart);
        new CChart(axis, numberOfDefects);
    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new CChartEx1());
    }
}
UChart Class

public class Imsl.Chart2D.QC.UChart : ShewhartControlChart

UChart is a u-chart for monitoring the defect rate when defects are rare.
The defect rate is the number of defects found divided by the number of samples inspected. The number of defects are assumed to be rare. Control limits are computed using the Poisson distribution. If defects are not rare, use PChart (p. 1556) instead. The sample sizes are not required to be equal.
The control limits are at

\[ \bar{u} + k \sqrt{\bar{u}/n} \]

where \( \bar{u} \) is the observed average number of defects per unit, \( n \) is the number of inspection units, and \( k \) is the value of the “ControlLimit” attribute for the line. By default, the chart contains an upper control limit line with \( k=3 \), a lower control limit line with \( k=-3 \), and a central line equal to \( \bar{u} \). Additional control limits can be added. The method AddWeco adds control limits with \( k = -2, -1, 1, 2 \).

### Constructors

**UChart**

public UChart(Imsl.Chart2D.AxisXY axis, double sizeSample, int[] numberDefects)

**Description**

Creates a u-Chart given the number of defects for a series of samples with equal sample sizes.

**Parameters**

- **axis** – The AxisXY parent of this node.
- **sizeSample** – The size of each sample.
- **numberDefects** – An array of arrays containing the number of defects. The number of defects must all be nonnegative.

**UChart**

public UChart(Imsl.Chart2D.AxisXY axis, double[] sizeSample, int[] numberDefects)

**Description**

Creates a u-Chart given the number of defects rates for a series of samples with varying sample sizes.

**Parameters**

- **axis** – The AxisXY parent of this node.
- **sizeSample** – An array containing the size of each sample.
- **numberDefects** – An array of arrays containing the number of defects. The number of defects must all be nonnegative. The length of the sizeSample and numberDefects arrays must be equal.

### Method

**PrePaint**

override public void PrePaint()

**Description**

Setup chart with current settings.
Example: u-Chart

The number of defects in each of 12 samples was counted. The number of items in the samples varied from 8 to 12.

```csharp
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class UChartEx1 : FrameChart
{
    static private readonly int[] numberDefects = {
        3, 4, 2, 3, 2, 1, 4, 2, 4, 3, 1, 1
    };
    static private readonly double[] sizeSample = {
        10, 8, 8, 10, 12, 12, 10, 10, 8, 8, 10, 12
    };

    public UChartEx1()
    {
        AxisXY axis = new AxisXY(this.Chart);
        axis.AxisY.AxisTitle.SetTitle("Number Defective");
        new UChart(axis, sizeSample, numberDefects);
    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new UChartEx1());
    }
}
```
public class Imsl.Chart2D.QC.EWMA : ShewhartControlChart

EWMA is an **exponentially weighted moving average** control chart.

The EWMA statistic is given by

\[ \text{EWMA}_t = \lambda x_t + (1 - \lambda) \text{EWMA}_{t-1} \]
where \( x_t \) is the observation at time \( t \) and \( 0 < \lambda \leq 1 \) is the decay parameter which determines the depth of memory of EWMA.

**Property**

**Lambda**

virtual public double Lambda {get; set; }

**Description**

The value of the attribute “Lambda”, the decay parameter.

**Remarks**

Its default value is 0.25.

**Constructors**

**EWMA**

public EWMA(Imsl.Chart2D.AxisXY axis, double[] x, double lambda)

**Description**

Creates an exponentially weighted moving average chart. The expected mean and standard deviation are computed from the sample data.

**Parameters**

- **axis** – The AxisXY parent of this node
- **x** – An array containing the data.
- **lambda** – The decay parameter. It is usually between 0.2 and 0.3.

**Exception**

System.ArgumentException is thrown if \( 0 < \lambda \leq 1 \) does not hold.

**EWMA**

public EWMA(Imsl.Chart2D.AxisXY axis, double[] data, double lambda, double expectedMean, double expectedStandardDeviation)

**Description**

Creates an exponentially weighted moving average chart using the given values for the expected mean and standard deviation.
Parameters

axis – The AxisXY parent of this node.
data – An array containing data.
lambda – The decay parameter. It is usually between 0.2 and 0.3.
expectedMean – The expected mean of the data.
expectedStandardDeviation – The expected standard deviation of the data.

Exception

System.ArgumentException is thrown if $0 < \lambda \leq 1$ does not hold.

Method

PrePaint
override public void PrePaint()

Description
Setup chart with current settings.

Example: EWMA Chart

A process is monitored using the EWMA control chart. The data is from NIST Engineering Statistics Handbook: EWMA Control Charts.

```csharp
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class EWMAEx1 : FrameChart
{
    static private readonly double[] data = {
        52.0, 47.0, 53.0, 49.3, 50.1, 47.0,
        51.0, 50.1, 51.2, 50.5, 49.6, 47.6,
        49.9, 51.3, 47.8, 51.2, 52.6, 52.4,
        53.6, 52.1
    };

    public EWMAEx1()
    {
        AxisXY axis = new AxisXY(this.Chart);
        double lambda = 0.3;
        EWMA ewma = new EWMA(axis, data, lambda);
    }

    public static void Main(string[] argv)
    {
```
CuSum Class

public class Imsl.Chart2D.QC.CuSum : ShewhartControlChart
CuSum is a *cumulative sum* chart. It is more efficient than a Shewhart chart for detecting small shifts in the mean of a process.

CuSum plots the cumulative sum of the deviations of the expected value. If $\mu_0$ is the expected mean for a process and $\bar{x}_i$ are the sample means then the cumulative sum is

$$C_i = C_{i-1} + (\bar{x}_i - \mu_0)$$

**Property**

**ExpectedMean**

```csharp
virtual public double ExpectedMean {get; set; }
```

**Description**
The expected mean of all of the data from all of the samples.

**Property Value**
The value of `ExpectedMean` is either the expected mean value or the target mean. Its default value is computed from the data passed to the constructor.

**Constructor**

```csharp
public CuSum(Imsl.Chart2D.AxisXY axis, double[] xbar)
```

**Description**
Creates a CuSum chart given the means of a series of samples.

**Parameters**
- `axis` – The `AxisXY` parent of this node
- `xbar` – The means of the samples.

**Methods**

**PrePaint**

```csharp
override public void PrePaint()
```
Description
Setup chart with current settings.

SetDataRange
override public void SetDataRange(double[] range)

Description
Update the data range, \(\text{range} = \{\text{xmin, xmax, ymin, ymax}\}\)

Parameter
\(\text{range}\) – A double array which contains the updated range, \(\{\text{xmin, xmax, ymin, ymax}\}\)

Remarks
The entries in range are updated to reflect the extent of the data in this node. The argument range is an input/output variable. Its value should be updated only if the data in this node is outside the range already in the array.

Example: CuSum Chart
A CuSum chart is constructed from the data at NIST Engineering Statistics Handbook: CuSum Control Charts.

```csharp
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

class CuSumEx1 : FrameChart
{
    static private readonly double[] data = {
        324.925, 324.675, 324.725, 324.350, 325.350, 325.225, 324.125,
        324.525, 325.225, 324.600, 324.625, 325.150, 328.325, 327.250,
        327.825, 328.500, 326.675, 327.775, 326.875, 328.350
    };
    public CuSumEx1()
    {
        AxisXY axis = new AxisXY(this.Chart);
        double mean = 325;
        CuSum cusum = new CuSum(axis, data);
        cusum.ExpectedMean = mean;
    }
    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new CuSumEx1());
    }
}
```

Quality Control and Improvement Charts CuSum • 1571
CuSumStatus Class

```
public class Imsl.Chart2D.QC.CuSumStatus : ShewhartControlChart

CuSumStatus is a cumulative sum status chart. CuSumStatus plots the tabular CuSum results.

The one-sided upper and lower cusums, $C^+$ and $C^-$ are computed. These are plotted as two bar charts. The $C^+$ are plotted as a bar chart above the x-axis and $C^-$ is below the axis. By default, in-control bars are green and out-of-control bars are red.
```
Control limit lines are drawn at ±h, where h is the value of the property AbsoluteH. This can be set either to an absolute number, using the property AbsoluteH, or relative to the standard deviation of the data, using the property RelativeH.

**Properties**

**AbsoluteH**
virtual public double AbsoluteH {get; set; }

**Description**
The value for H used for setting limits.

**Property Value**
There is no attribute “AbsoluteH”. Its value is Sigma times RelativeH.

**BarMinus**
virtual public Imsl.Chart2D.Bar BarMinus {get; }

**Description**
The value of the attribute “BarMinus” contains the Bar (p. 1464) object for the negative CuSum (C⁻) bars.

**Property Value**
To access the C⁻ bars for where the process is in control, apply the method GetBarSet(0,0) to the Bar object returned by this method. For the bars for where the process is not in control, apply the method GetBarSet(1,0) to the Bar object returned by this method.

**BarPlus**
virtual public Imsl.Chart2D.Bar BarPlus {get; }

**Description**
The value of the attribute “BarPlus” contains the Bar (p. 1464) object for the positive CuSum (C⁺) bars.

**Property Value**
To access the C⁺ bars for where the process is in control, apply the method GetBarSet(0,0) to the Bar object returned by this method. For the bars for where the process is not in control, apply the method GetBarSet(1,0) to the Bar object returned by this method.

**CMinus**
virtual public double[] CMinus {get; }

**Description**
The C⁻ values.

**CPlus**
virtual public double[] CPlus {get; }

---

**Quality Control and Improvement Charts**

CuSumStatus • 1573
Description
The $C^+$ values.

**DataMarkers**

virtual public Imsl.Chart2D.Data DataMarkers {get; }

Description
The “DataMarkers” attribute containing the data markers.

**DataMarkersAxis**

virtual public Imsl.Chart2D.AxisXY DataMarkersAxis {get; }

Description
The “DataMarkersAxis” attribute containing the axis associated with the data markers. This is normally the right-side axis.

**ExpectedMean**

virtual public double ExpectedMean {get; set; }

Description
The expected mean of all of the data from all of the samples.

**Property Value**

The value of ExpectedMean is either the expected mean value or the target mean. Its default value is computed from the data passed to the constructor.

**InitialCMinus**

virtual public double InitialCMinus {get; set; }

Description
The initial value of $C^-$. 

**Property Value**

This is used to initiate a fast initial response (headstart). Typically, this would be set to $H/2$ for a 50% headstart. Its default value is zero.

**InitialCPlus**

virtual public double InitialCPlus {get; set; }

Description
The initial value of $C^+$. 

**Property Value**

This is used to initiate a fast initial response (headstart). Typically, this would be set to $H/2$ for a 50% headstart. Its default value is zero.

**NMinus**

virtual public int[] NMinus {get; }

1574 • CuSumStatus
IMSL C# Numerical Library
Description

$N^{-}$, the number of consecutive periods that the cusums $C_{i}^{-}$ have been nonzero.

**NPlus**

virtual public int[] NPlus {get; }

Description

$N^{+}$, the number of consecutive periods that the cusums $C_{i}^{+}$ have been nonzero.

**RelativeH**

virtual public double RelativeH {get; set; }

Description

The relative $h$.

**Property Value**

This is the value of the attribute “RelativeH”. Its default value is 3.0.

**Sigma**

virtual public double Sigma {get; set; }

Description

The standard deviation of the data.

**SlackValue**

virtual public double SlackValue {get; }

Description

The slack value. This is also called $K$, or the reference value or the allowance.

**Constructor**

**CuSumStatus**

public CuSumStatus(Imsl.Chart2D.AxisXY axis, double[] data, double expectedMean, double slackValue)

Description

Creates a CuSumStatus chart.

**Parameters**

- axis – The AxisXY parent of this node
- data – An array of measurements of a continuous variable.
- expectedMean – The expected (or target) mean.
- slackValue – The slack value, also called $K$, or the reference value, or the allowance.
Methods

AddDataMarkers
virtual public Imsl.Chart2D.Data AddDataMarkers()

Description
Adds the original data to the chart on a newly created axis. The new axis is stored as attribute “DataMarkersAxis” and the new Data node is stored as attribute “DataMarkers”.

Returns
A Data object containing the markers.

AddDataMarkers
virtual public Imsl.Chart2D.Data AddDataMarkers(Imsl.Chart2D.AxisXY axisDataMarkers)

Description
Adds the original data to the chart. The axis is stored as attribute “DataMarkersAxis” and the new Data node is stored as attribute “DataMarkers”.

Parameter
axisDataMarkers – The axis for the data markers.

Returns
A Data object containing the markers.

CreateDataAxis
virtual public Imsl.Chart2D.AxisXY CreateDataAxis()

Description
Creates a new axis to hold a cummulative line. The created axis is drawn on the right side. Its x-axis is not drawn, since it would overlap with the primary chart axis. The x-axis is set to have the same window size as the orignal axis and its autoscaling attribute is turned off.

Returns
An AxisXY containing the newly created axis.

PrePaint
override public void PrePaint()

Description
Setup chart with current settings.

Print
virtual public void Print()
**Description**

Prints the tabular CuSum results. This prints the input data, $C^+$, $N^+$, $C^-$, and $N^-$.  

**SetX**

override public void SetX(double[] x)

---

**Description**

Sets the x-coordinates of the bars. This also sets the “BarWidth” attribute to $(x[1] - x[0]) / 2$.

**Parameter**

x – An array containing the x-coordinates. Its length must equal the length of the data array used to construct this object.

---

**Example: CuSumStatus Chart**

A process is monitored using the CuSumStatus control chart. The solid red bars indicate the process is out-of-control. The hollow green bars indicate that the process is in control. The data is from NIST Engineering Statistics Handbook: CuSumStatus Control Charts.

```csharp
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

class CuSumStatusEx1 : FrameChart
{
    static private readonly double[] data = {52.0, 47.0, 53.0, 49.3, 50.1, 47.0,
                                              51.0, 50.1, 51.2, 50.5, 49.6, 47.6,
                                              49.9, 51.3, 47.8, 51.2, 52.6, 52.4,
                                              53.6, 52.1};

    public CuSumStatusEx1()
    {
        AxisXY axis = new AxisXY(this.Chart);
        double mean = Imsl.Stat.Summary.GetMean(data);
        double slackValue = 0.5;
        CuSumStatus cusum = new CuSumStatus(axis, data, mean, slackValue);
        cusum.BarPlus.GetBarSet(0,0).FillType = ChartNode.FILL_TYPE_NONE;
        cusum.BarMinus.GetBarSet(0,0).FillType = ChartNode.FILL_TYPE_NONE;
    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new CuSumStatusEx1());
    }
}
```

---

**Quality Control and Improvement Charts**

CuSumStatus • 1577
ParetoChart Class

public class Imsl.Chart2D.QC.ParetoChart : Bar

ParetoChart is a Pareto bar chart.

The bars are sorted into descending order. It is used in quality assurance tracking to identify and prioritize areas of greatest impact. It extends Bar.
The method AddCumulativeLine adds a cumulative percentage line to the chart. This is the percent of defects accounted for by the current item and items to its left. If the cumulative percentage line is added, a second axis is created on the right. This is required because the units for this line are 0% to 100%. The units of the original axis (on the left) are the number of defects.

**Properties**

**CumulativeAxis**

```csharp
virtual public Imsl.Chart2D.AxisXY CumulativeAxis {get; }
```

**Description**
The “CumulativeAxis” attribute, the axis on which the cumulative line is drawn.

**CumulativeLine**

```csharp
virtual public Imsl.Chart2D.Data CumulativeLine {get; }
```

**Description**
The “CumulativeLine” attribute.

**Property Value**
The Data object containing the cumulative line.

**Constructors**

**ParetoChart**

```csharp
public ParetoChart(Imsl.Chart2D.AxisXY axisBar, string[] labels, int[] numberDefects)
```

**Description**
Constructs a Pareto chart.

**Parameters**
- `axisBar` – The AxisXY parent of this node. Its formatting is changed to integer formatting.
- `labels` – A String array which contains the labels for the data values.
- `numberDefects` – An int array which contains the number of defects. These data values must be in the same order as the values in labels, but they do not need to be sorted.

**Exception**
- `System.ArgumentException` is thrown if the length of the labels and numberDefects arrays are unequal or if any element of numberDefects is negative.
ParetoChart

```csharp
public ParetoChart(Imsl.Chart2D.AxisXY axisBar, string[] labels, int[] numberDefects, double maximumFractionCategoriesPlotted, string otherLabel)
```

**Description**
Constructs a Pareto chart showing only the most important bars.

**Parameters**
- **axisBar** – The AxisXY parent of this node. Its formatting is changed to integer formatting.
- **labels** – A String array which contains the labels for the data values.
- **numberDefects** – An int array which contains the number of defects. These data values must be in the same order as the values in labels, but they do not need to be sorted.
- **maximumFractionCategoriesPlotted** – The maximum cumulative fraction to be represented in separate categories. The remaining categories are consolidated into a single bar. A typical value for this argument is 0.80. This must be at least 0 and no more than 1.
- **otherLabel** – The label of the bar holding total defect count of the categories not plotted.

**Exception**
- `System.ArgumentException` is thrown if the length of the labels and numberDefects arrays are unequal or if any element of numberDefects is negative.

---

ParetoChart

```csharp
public ParetoChart(Imsl.Chart2D.AxisXY axisBar, string[] labels, int[] numberDefects, int maximumCategoriesPlotted, string otherLabel)
```

**Description**
Constructs a Pareto chart showing only a limited number of bars. The numberDefects-axis formatting is changed to integer formatting.

**Parameters**
- **axisBar** – The AxisXY parent of this node.
- **labels** – A String array which contains the labels for the data values.
- **numberDefects** – An int array which contains the number of defects. These data values must be in the same order as the values in labels, but they do not need to be sorted.
- **maximumCategoriesPlotted** – The maximum number of categories to be plotted. Categories with smaller number of defects are consolidated into a single bar. The total number of bars will be maximumCategoriesPlotted+1. This must be at least 0 and no more than the length of numberDefects.
- **otherLabel** – The label of the bar holding total defect count of the categories not plotted.

**Exception**
- `System.ArgumentException` is thrown if the length of the labels and numberDefects arrays are unequal or if any element of numberDefects is negative.
Methods

**AddCumulativeLine**
virtual public Imsl.Chart2D.Data AddCumulativeLine(Imsl.Chart2D.AxisXY axisCumulativeLine)

**Description**
Adds a cumulative line to the specified axis.

**Parameter**
- axisCumulativeLine – The axis for the cumulative line.

**AddCumulativeLine**
virtual public Imsl.Chart2D.Data AddCumulativeLine()

**Description**
Creates a new right-side axis and adds a cumulative line to it.

**Returns**
A Data object containing the cumulative percentage line.

**CreateCumulativeLineAxis**
virtual public Imsl.Chart2D.AxisXY CreateCumulativeLineAxis()

**Description**
Creates a new axis to hold a cumulative line.

**Returns**
An AxisXY object containing the newly created axis.

**Remarks**
The created axis is drawn on the right side. Its x-axis is not drawn, since it would overlap with the primary Pareto chart axis. Its y-axis is labeled using the “P0” (percent) format. Its y-axis is scaled to [0,1], it is not autoscaled.

**Example: Pareto Chart**
The number of defects caused by four different factors were measured and plotted as a Pareto chart. The class ParetoChart sorts the factors in order of number of defects.

```csharp
using Imsl.Chart2D;
using Imsl.Chart2D.QC;
using System;

public class ParetoEx1 : FrameChart
{
    public ParetoEx1()
    {
        AxisXY axisBar = new AxisXY(this.Chart);
```
int[] numberDefects = new int[]{71, 23, 128, 37};
ParetoChart pareto = new ParetoChart(axisBar, labels, numberDefects);
pareto.AddCumulativeLine();
Data cumulativeLine = pareto.CumulativeLine;
cumulativeLine.MarkerType = Data.MARKER_TYPE_FILLED_CIRCLE;
this.Chart.ChartTitle.SetTitle("Manufacturing Defects");
axisBar.AxisY.AxisTitle.SetTitle("Defects");
pareto.CumulativeAxis.AxisY.AxisTitle.SetTitle("Cumulative Defect Percentage");

public static void Main(string[] argv)
{
    System.Windows.Forms.Application.Run(new ParetoEx1());
}
}
Chapter 25. Quality Control and Improvement Charts

Pareto Chart:

Manufacturing Defects

- Relay: 120 defects
- Gear: 60 defects
- Spindle: 30 defects
- Diode: 15 defects

Cumulative Defect Percentage:
- 0%: 15 defects
- 25%: 30 defects
- 50%: 60 defects
- 75%: 90 defects
- 100%: 120 defects

Output
Chapter 26: Data Mining

Type

class NaiveBayesClassifier ................................................................. 1587

Usage Notes

Data Mining - An Overview

The Data mining process involves the use of a collection of statistical and analytical methods for extracting useful information from large databases. The problem of extracting information from large databases is common to government, industry, engineering and sciences.

Data Filtering

The first step in this process is to filter data from its raw form into formats required by sophisticated analytical algorithms.

Data fall into two major categories: continuous and categorical. Many algorithms, such as neural network forecasting, perform better if continuous data are mapped into a common scale. Class ScaleFilter implements several techniques for automatically scaling continuous data, including several variations of z-score scaling. If the continuous data represent a time series, TimeSeriesFilter and TimeSeriesClassFilter can be used to create a matrix of lagged values required as input to forecasting neural networks.

Categorical data must also be mapped into a corresponding numerical representation before they can be used in solving forecasting and classification problems. There are two types of categorical data: ordinal and nominal. Ordinal data have a natural ordering among the categories, such as a school grade. Nominal data are categories without a natural ordering, such as eye color. The function imsls_unsupervised_ordinal_filter encodes and decodes ordinal data into the range [0, 1] using cumulative percentages. The function imsls_unsupervised_nominal_filter uses binary encoding to map nominal data into a matrix of zeros and ones.

Naive Bayes

Naive Bayes is a classification algorithm. A classifier is trained using NaiveBayesClassifier with known classifications. Once the classifier is trained, the classifier can be used to classify patterns with
unknown classifications.

Classification problems can be solved using other algorithms such as discriminant analysis and neural networks. In general these alternatives have smaller classification error rates, but they are too slow for large classification problems. During training, NaiveBayesClassifier uses the non-missing training data to estimate two-way correlations among the attributes. Higher order correlations are assumed to be zero. This can increase the classification error rate, but it significantly reduces the time needed to train the classifier.

In addition, the Naive Bayes algorithm is the only classification algorithm that can handle data with missing values. Other algorithms such as discriminant analysis do not allow missing values in the data. This is a significant limitation for applying other techniques to a larger database.

Classification problems are characterized by a need to classify unknown patterns or data into one of m categories based upon the values of k attributes $x_1, x_2, \ldots, x_k$. There are many algorithms for solving classification problems including discriminant analysis, neural networks and Naive Bayes. Each algorithm has its strengths and weaknesses. Discriminant analysis is robust but it requires $x_1, x_2, \ldots, x_k$ to be continuous, and since it uses a simple linear equation for the discriminant function, its error rate can be higher than the other algorithms. See IMSL.Stat.DiscriminantAnalysis.

Neural Networks provides a linear or non-linear classification algorithm that accepts both nominal and continuous input attributes. However, network training can be unacceptably slow for problems with a larger number of attributes, typically when $k > 1000$. Naive Bayes, on the other hand, is a simple algorithm that is very fast. A Naive Bayes classifier can be trained to classify patterns involving thousands of attributes and applied to thousands of patterns. As a result, Naive Bayes is a preferred algorithm for text mining and other large classification problems. However, its computational efficiency comes at a price. The error rate for a Naive Bayes classifier is typically higher than the equivalent Neural Network classifier, although it is usually low enough for many applications such as text mining.

If $C$ is the classification attribute and $X^T = x_1, x_2, \ldots, x_k$ is the vector valued array of input attributes, the classification problem simplifies to estimating the conditional probability $P(c|X)$ from a set of training patterns. The Bayes rule states that this probability can be expressed as the ratio:

$$P(C = c|X = \{x_1, x_2, \ldots, x_k\}) = \frac{P(C = c)P(X = \{x_1, x_2, \ldots, x_k\}|C = c)}{P(X = \{x_1, x_2, \ldots, x_k\})}$$

where $c$ is equal to one of the target classes $0, 1, \ldots, n\text{Classes} - 1$. In practice, the denominator of this expression is constant across all target classes since it is only a function of the given values of $X$. As a result, the Naive Bayes algorithm does not expend computational time estimating $P(X = \{x_1, x_2, \ldots, x_k\})$ for every pattern. Instead, a Naive Bayes classifier calculates the numerator $P(C = c)P(X = \{x_1, x_2, \ldots, x_k\})$ for each target class and then classifies $X$ to the target class with the largest value, i.e.,

$$X \leftarrow \max_{c=0,1,\ldots,n\text{Classes}-1} P(C = c)P(X|C = c)$$

This is equivalent to assuming that the values of the input attributes, given $C$, are independent of one another, i.e.

$$P(x_i|x_j, C = c) = P(x_i|C = c), \text{ for all } i \neq j$$

In real world data this assumption rarely holds, yet in many cases this approach results in surprisingly low classification error rates. Since, the estimate of $P(C = c|X = \{x_1, x_2, \ldots, x_k\})$ from a Naive Bayes
classifier is generally an approximation, classifying patterns based upon the Naive Bayes algorithm can have acceptably low classification error rates.

Class NaiveBayesClassifier is used to train a classifier from a set of training patterns that contains patterns with values for both the input and target attributes. Classifications of new patterns with unknown classifications can be predicted using NaiveBayesClassifier methods PredictClass and Probabilities.

Neural Networks

Neural networks can be used for a variety of problems, some of which have been solved by existing statistical methods, and some of which have not. These applications fall into one of three categories, forecasting, classification, or statistical pattern recognition. More details on Neural Networks can be found in the Neural Networks chapter.

NaiveBayesClassifier Class

public class Imsl.DataMining.NaiveBayesClassifier

Trains a Naive Bayes Classifier

NaiveBayesClassifier trains a Naive Bayes classifier for classifying data into one of nClasses target classes. Input attributes can be a combination of both nominal and continuous data. Ordinal data can be treated as either nominal attributes or continuous. If the distribution of the ordinal data is known or can be approximated using one of the continuous distributions, then associating them with continuous attributes allows a user to specify that distribution. Missing values are allowed.

Before training the classifier the input attributes must be specified. For each nominal attribute, use method CreateNominalAttribute to specify the number of categories in each nNominal attribute. Specify the input attributes in the same column order that they will be supplied to the Train method. For example, if the input attribute in the first two columns of the nominal input data, nominalData, represent the first two nominal attributes and have two and three categories respectively, then the first call to the CreateNominalAttribute method would specify two categories and the second call to CreateNominalAttribute would specify three categories.

Likewise, for each continuous attribute, the method CreateContinuousAttribute can be used to specify a IProbabilityDistribution other than the default NormalDistribution. A second CreateContinuousAttribute is provided to allow specification of a different distribution for each target class (see Example 3). Create each continuous attribute in the same column order they will be supplied to the Train method. If CreateContinuousAttribute is not invoked for all nContinuous attributes, the NormalDistribution probability distribution will be used. For example, if five continuous attributes have been specified in the constructor, but only three calls to CreateContinuousAttribute have been invoked, the last two attributes, or columns of continuousData in the Train method, will use the NormalDistribution probability distribution.

Nominal only, continuous only, and a combination of both nominal and continuous input attributes are allowed. The Train method allows either nominal or continuous input arrays to be null.
Let $C$ be the classification attribute with target categories $0, 1, \ldots, n\text{Classes} - 1$, and let $X = \{x_1, x_2, \ldots, x_k\}$ be a vector valued array of $k = n\text{Nominal} + n\text{Continuous}$ input attributes, where $n\text{Nominal}$ is the number of nominal attributes and $n\text{Continuous}$ is the number of continuous attributes. See methods CreateNominalAttribute to specify the number of categories for each nominal attribute and CreateContinuousAttribute to specify the distribution for each continuous attribute. The classification problem simplifies to estimate the conditional probability $P(C|X)$ from a set of training patterns. The Bayes rule states that this probability can be expressed as the ratio:

$$P(C = c|X = \{x_1, x_2, \ldots, x_k\}) = \frac{P(C = c)P(X = \{x_1, x_2, \ldots, x_k\}|C = c)}{P(X = \{x_1, x_2, x, \ldots, x_k\})}$$

where $c$ is equal to one of the target classes $0, 1, \ldots, n\text{Classes} - 1$. In practice, the denominator of this expression is constant across all target classes since it is only a function of the given values of $X$. As a result, the Naive Bayes algorithm does not expend computational time estimating $P(X = \{x_1, x_2, x, \ldots, x_k\})$ for every pattern. Instead, a Naive Bayes classifier calculates the numerator $P(C = c)P(X = \{x_1, x_2, \ldots, x_k\}|C = c)$ for each target class and then classifies $X$ to the target class with the largest value, i.e.,

$$X \leftarrow \max_{c=0,1,\ldots,n\text{Classes}-1} P(C = c)P(X|C = c)$$

The classifier simplifies this calculation by assuming conditional independence. That is it assumes that:

$$P(X = \{x_1, x_2, \ldots, x_k\}|C = c) = \prod_{j=1}^{k} P(x_j|C = c)$$

This is equivalent to assuming that the values of the input attributes, given $C$, are independent of one another, i.e.,

$$P(x_i|x_j, C = c) = P(x_i|C = c), \text{ for all } i \neq j$$

In real world data this assumption rarely holds, yet in many cases this approach results in surprisingly low classification error rates. Since, the estimate of $P(C = c|X = \{x_1, x_2, \ldots, x_k\})$ from a Naive Bayes classifier is generally an approximation, classifying patterns based upon the Naive Bayes algorithm can have acceptably low classification error rates.

For nominal attributes, this implementation of the Naive Bayes classifier estimates conditional probabilities using a smoothed estimate:

$$P(x_j|C = c) = \frac{N_{\{x_j \cap C = c\}} + \hat{\lambda}}{N_{\{C = c\}} + \hat{\lambda} n_j},$$

where $N\{Z\}$ is the number of training patterns with attribute $Z$ and $j$ is equal to the number of categories associated with the $j$-th attribute.

The probability $P(C = c)$ is also estimated using a smoothed estimate:

$$P(C = c) = \frac{N\{C = c\} + \hat{\lambda}}{n\text{Patterns} + \hat{\lambda} (n\text{Classes})}.$$
smoothing. The default smoothing used in this algorithm, $\lambda = 1$, is commonly referred to as Laplace smoothing. This can be specified using the property DiscreteSmoothingValue.

For continuous attributes, the same conditional probability $P(x_j|C = c)$ in the Naive Bayes formula is replaced with the conditional probability density function $f(x_j|C = c)$. By default, the density function for continuous attributes is the normal (Gaussian) probability density function (see NormalDistribution):

$$f(x_j|C = c) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x_j - \mu)^2}{2\sigma^2}}$$

where $\mu$ and $\sigma$ are the conditional mean and standard deviation, i.e. the mean and standard deviation of $x_j$ when $C = c$. For convenience, methods GetMeans and GetStandardDeviations are provided to calculate the conditional mean and standard deviations of the training patterns.

In addition to the default normal pdf, users can select any continuous distribution to model the continuous attribute by providing an implementation of the Imsl.Stat.IProbabilityDistribution interface. See NormalDistribution, LogNormalDistribution, GammaDistribution, and PoissonDistribution for classes that implement the IProbabilityDistribution interface.

Smoothing conditional probability calculations for continuous attributes is controlled by the properties ContinuousSmoothingValue and ZeroCorrection. By default, conditional probability calculations for continuous attributes are unadjusted for calculations near zero. The value specified in the ContinuousSmoothingValue property will be added to each continuous probability calculation. This is similar to the effect of setting the property DiscreteSmoothingValue for the corresponding discrete calculations.

The value specified in the ZeroCorrection property is used when $(f(x|C = c) + \lambda) = 0$, where $\lambda$ is the smoothing parameter setting. If this condition occurs, the conditional probability is replaced with the property value set in ZeroCorrection.

Methods GetClassificationErrors, GetPredictedClass, GetProbabilities, and GetTrainingErrors provide information on how well the trained NaiveBayesClassifier predicts the known target classifications of the training patterns.

Methods Probabilities and PredictClass estimate classification probabilities and predict classification of the input pattern using the trained Naive Bayes Classifier. The predicted classification returned by PredictClass is the class with the largest estimated classification probability. Method ClassError predicts the classification from the trained Naive Bayes classifier and compares the predicted classifications with the known target classification provided. This allows verification of the classifier with a set of patterns other than the training patterns.

Properties

ContinuousSmoothingValue

public double ContinuousSmoothingValue {get; set; }

Description

Parameter for calculating smoothed estimates of conditional probabilities for continuous attributes.
Property Value
A double containing the smoothing parameter to be used for calculating smoothed estimates of conditional probabilities for continuous attributes. ContinuousSmoothingValue must be non-negative.
Default: ContinuousSmoothingValue=0, i.e. no smoothing is done.

DiscreteSmoothingValue
public double DiscreteSmoothingValue {get; set; }

Description
Parameter for calculating smoothed estimates of conditional probabilities for discrete (nominal) attributes.

Property Value
A double containing the smoothing parameter to be used for calculating smoothed estimates of conditional probabilities for discrete attributes. DiscreteSmoothingValue must be non-negative.
Default: DiscreteSmoothingValue = 1.0, i.e. Laplace smoothing of conditional probabilities.

ZeroCorrection
public double ZeroCorrection {get; set; }

Description
Specifies the replacement value to be used for conditional probabilities equal to zero.

Property Value
A double containing the value to replace conditional probabilities equal to zero. ZeroCorrection must be non-negative.
Default: No correction will be performed.

Constructor

NaiveBayesClassifier
public NaiveBayesClassifier(int nContinuous, int nNominal, int nClasses)

Description
Constructs a NaiveBayesClassifier.

Parameters
nContinuous – An int containing the number of continuous attributes.
nNominal – An int containing the number of nominal attributes.
nClasses – An int containing the number of target classifications.
Methods

**ClassError**

```java
public double ClassError(double[] continuous, int[] nominal, int classification)
```

**Description**

Returns the classification probability error for the input pattern and known target classification.

**Parameters**

- `continuous` – A `double` array of length `nContinuous` containing an input pattern of continuous attributes. If `nContinuous = 0`, a null is allowed.
- `nominal` – An `int` array of length `nNominal` containing an input pattern of nominal attributes. If `nNominal = 0`, a null is allowed.
- `classification` – An `int` containing the target classification.

**Returns**

A `double` containing the classification probability error for the input pattern. The classification error for the input pattern is equal to $1 - p$, where $p$ is the predicted class probability of input `classification`. The predicted class probability of input `classification` can be obtained by the method `Probabilities`. If $p = Probabilities$ and $k$ is equal to `classification`, then the classification error is $1 - p[k]$.

**CreateContinuousAttribute**

```java
public void CreateContinuousAttribute(Imsl.Stat.IProbabilityDistribution pdf)
```

**Description**

Create a continuous variable and the associated distribution function.

**Parameter**

- `pdf` – A `IProbabilityDistribution` to be applied to the continuous attribute. The distribution function will be applied to all classes. Default: `NormalDistribution` is used.

**CreateContinuousAttribute**

```java
public void CreateContinuousAttribute(Imsl.Stat.IProbabilityDistribution[] pdf)
```

**Description**

Create a continuous variable and the associated distribution functions for each target classification.

**Parameter**

- `pdf` – An array of `IProbabilityDistributions` containing `nClasses` distribution functions for a continuous attribute. This allows a different distribution function to be applied to each classification. Default: `NormalDistribution` is used.

**CreateNominalAttribute**

```java
public void CreateNominalAttribute(int nCategories)
```

**Data Mining**

**NaiveBayesClassifier** • 1591
Description
Create a nominal attribute and the number of categories

Parameter
- nCategories – An int containing the number of categories in the nominal attribute. The category
  values are expected to be encoded with integers ranging from 0 to nCategories -1. No default is
  used for nCategories. If nNominal is not zero, and CreateNominalAttribute is not invoked
  for each nNominal attribute, an InvalidOperationException will be thrown when the Train
  method is invoked.

GetClassCounts
public int[] GetClassCounts(int[] classificationData)

Description
Returns the number of patterns for each target classification.

Parameter
- classificationData – An int array containing the target classifications for the training
  patterns. These must be encoded from zero to nClasses -1. Any value outside this range is
  considered a missing value. In this case, the data in that pattern are not used to train the Naive
  Bayes classifier. However, any pattern with missing values is still classified after the classifier is
  trained.

Returns
An int array containing the class counts.

GetClassificationErrors
public double[] GetClassificationErrors()

Description
Returns the classification probability errors for each pattern in the training data.

Returns
A double array containing the classification probability errors for each pattern in the training data. The
classification error for the i-th training pattern is equal to 1 - predictedClassProbability[i][k], where
predictedClassProbability is returned from GetProbabilities and k is equal to
classificationData[i].

GetMeans
public double[][] GetMeans(double[][] continuousData, int[] classificationData)

Description
Returns a table of means for each continuous attribute in continuousData segmented by the target
classes in classificationData.

Parameters
- continuousData – A double matrix containing training values for the continuous attributes.
- classificationData – An int array containing the target classifications for the training
  patterns.
Returns
A continuousData[0].Length by nClasses double matrix, means, containing the means segmented by the target classes. The i-th row contains the means of the i-th continuous attribute for each value of the target classification. That is, means[i][j] is the mean for the i-th continuous attribute when the target classification equals j, unless there are no training patterns for this condition.

Remarks
This method is provided as a utility, prior training is not necessary.

GetPredictedClass
public int[] GetPredictedClass()

Description
Returns the predicted classification for each training pattern.

Returns
An int array containing the predicted classification for each training pattern.

GetProbabilities
public double[][] GetProbabilities()

Description
Returns the predicted classification probabilities for each target class.

Returns
A double matrix, prob, of size nPatterns by nClasses containing the predicted classification probabilities for each target class, where nPatterns is the number of patterns trained. prob[i][j] is the estimated probability that the i-th pattern belongs to the j-th target class.

GetStandardDeviations
public double[][] GetStandardDeviations(double[][] continuousData, int[] classificationData)

Description
Returns a table of standard deviations for each continuous attribute in continuousData segmented by the target classes in classificationData.

Parameters
    continuousData – A double matrix containing training values for the continuous attributes.
    classificationData – An int array containing the target classifications for the training patterns.

Returns
A continuousData[0].Length by nClasses double matrix, stdev, containing the standard deviations segmented by the target classes. The i-th row contains the standard deviation of the i-th continuous attribute for each value of the target classification. That is, stdev[i][j] is the standard deviations for the i-th continuous attribute when the target classification equals j, unless there are no training patterns for this condition.
Remarks
This method is provided as a utility, prior training is not necessary.

GetTrainingErrors
public int[][] GetTrainingErrors()

Description
Returns a table of classification errors of non-missing classifications for each target classification plus the overall total of classification errors.

Returns
An int matrix containing nClasses + 1 rows and two columns. The first column contains the number of misclassifications and the second column contains the total number of classifications for the i-th row target class. The last row of the matrix contains the total number of misclassifications in column one and the total non-missing classifications in column two.

IgnoreMissingValues
public void IgnoreMissingValues(bool ignoreMissing)

Description
Specifies whether or not missing values will be ignored during the training process.

Parameter
ignoreMissing – A boolean specifying whether or not to ignore patterns during training when one or more input attributes are missing. By default, both missing and non-missing values are used to train the classifier. Classification predictions are still returned for all patterns even when set to true.
Default: ignoreMissing = false.

PredictClass
public int PredictClass(double[] continuous, int[] nominal)

Description
Predicts the classification for the input pattern using the trained Naive Bayes classifier.

Parameters
continuous – A double array containing an input pattern of nContinuous continuous attributes. If nContinuous = 0, a null is allowed.
nominal – An int array of length nNominal containing an input pattern of nominal attributes. If nNominal = 0, a null is allowed.

Returns
An int containing the predicted classification for the input pattern using the trained Naive Bayes Classifier. The predicted classification returned is the class with the largest estimated classification probability. The classification probabilities can be predicted using the Probabilities method.

Probabilities
public double[] Probabilities(double[] continuous, int[] nominal)
**Description**
Predicts the classification probabilities for the input pattern using the trained Naive Bayes classifier.

**Parameters**
- `continuous` – A `double` array containing an input pattern of `nContinuous` continuous attributes. If `nContinuous` = 0, a null is allowed.
- `nominal` – An `int` array of length `nNominal` containing an input pattern of nominal attributes. If `nNominal` = 0, a null is allowed.

**Returns**
a `double` array of length `nClasses` containing the predicted classification probabilities for each target class.

**Train**
```
public void Train(double[][] continuousData, int[][] nominalData, int[] classificationData)
```

**Description**
Trains a Naive Bayes classifier for classifying data into one of `nClasses` target classifications.

**Parameters**
- `continuousData` – A `double` matrix containing the training values for the `nContinuous` continuous attributes. The `i`-th row contains the input attributes for the `i`-th training pattern. The `j`-th column contains the values for the `j`-th continuous attribute. Missing values should be set to `Double.NaN`. Patterns with both non-missing and missing values are used to train the classifier unless the `IgnoreMissingValues` method has been set to `true`. A null is allowed when `nContinuous` is equal to zero.
- `nominalData` – An `int` matrix containing the training values for the `nNominal` nominal attributes. The `i`-th row contains the input attributes for the `i`-th training pattern. The `j`-th column contains the classifications for the `j`-th nominal attribute. The values for the `j`-th nominal attribute are expected to be encoded with integers starting from 0 to `nCategories - 1`, where `nCategories` is specified in the `CreateNominalAttribute` method. Any value outside this range is treated as a missing value. Patterns with both non-missing and missing values are used to train the classifier unless the `IgnoreMissingValues` method has been set to `true`. A null is allowed when `nNominal` is equal to zero.
- `classificationData` – An `int` array containing the target classifications for the training patterns. These must be encoded from zero to `nClasses - 1`. Any value outside this range is considered a missing value. In this case, the data in that pattern are not used to train the Naive Bayes classifier. However, any pattern with missing values is still classified after the classifier is trained.

**Example 1: Continuous Attribute Example**
Fisher’s (1936) Iris data is often used for benchmarking classification algorithms. It consists of the following continuous input attributes and a classification target:

- **Continuous Attributes Usage**

---

**Data Mining**

**NaiveBayesClassifier • 1595**
– Sepal Length
– Sepal Width
– Petal Length
– Petal Width

• Classification of Iris Type
  – Setosa
  – Versicolour
  – Virginica

This example trains a Naive Bayes classifier using 140 of the 150 continuous patterns, then classifies ten unknown plants using their sepal and petal measurements.

```csharp
using System;
using Imsl.DataMining;

public class NaiveBayesClassifierEx1
{
    private static double[,] irisFisherData = new double[,]{
        {1.0, 5.1, 3.5, 1.4, .2},
        {1.0, 4.9, 3.0, 1.4, .2},
        {1.0, 4.7, 3.2, 1.3, .2},
        {1.0, 4.6, 3.1, 1.5, .2},
        {1.0, 5.0, 3.6, 1.4, .2},
        {1.0, 5.4, 3.9, 1.7, .4},
        {1.0, 4.6, 3.4, 1.4, .3},
        {1.0, 5.0, 3.4, 1.5, .2},
        {1.0, 4.4, 2.9, 1.4, .2},
        {1.0, 4.9, 3.1, 1.5, .1},
        {1.0, 5.4, 3.7, 1.5, .2},
        {1.0, 4.8, 3.4, 1.6, .2},
        {1.0, 4.8, 3.0, 1.4, .1},
        {1.0, 4.3, 3.0, 1.1, .1},
        {1.0, 5.8, 4.0, 1.2, .2},
        {1.0, 5.7, 4.4, 1.5, .4},
        {1.0, 5.4, 3.9, 1.3, .4},
        {1.0, 5.1, 3.5, 1.4, .3},
        {1.0, 5.7, 3.8, 1.7, .3},
        {1.0, 5.1, 3.8, 1.5, .3},
        {1.0, 5.4, 3.4, 1.7, .2},
        {1.0, 5.1, 3.7, 1.5, .4},
        {1.0, 4.6, 3.6, 1.0, .2},
        {1.0, 5.1, 3.3, 1.7, .5},
        {1.0, 4.8, 3.4, 1.9, .2},
        {1.0, 5.0, 3.0, 1.6, .2},
        {1.0, 5.0, 3.4, 1.6, .4},
        {1.0, 5.2, 3.5, 1.5, .2},
        {1.0, 5.2, 3.4, 1.4, .2},
        {1.0, 4.7, 3.2, 1.6, .2},
        {1.0, 4.8, 3.1, 1.6, .2},
        {1.0, 5.4, 3.4, 1.5, .4},
```
new double[]{1.0, 5.2, 4.1, 1.5, .1},
new double[]{1.0, 5.5, 4.2, 1.4, .2},
new double[]{1.0, 4.9, 3.1, 1.5, .2},
new double[]{1.0, 5.0, 3.2, 1.2, .2},
new double[]{1.0, 5.5, 3.5, 1.3, .2},
new double[]{1.0, 4.9, 3.6, 1.4, .1},
new double[]{1.0, 4.4, 3.0, 1.3, .2},
new double[]{1.0, 5.1, 3.4, 1.5, .2},
new double[]{1.0, 5.0, 3.5, 1.3, .2},
new double[]{1.0, 4.5, 2.3, 1.3, .3},
new double[]{1.0, 4.4, 3.2, 1.3, .2},
new double[]{1.0, 5.0, 3.5, 1.6, .6},
new double[]{1.0, 5.1, 3.8, 1.9, .4},
new double[]{1.0, 4.8, 3.0, 1.4, .3},
new double[]{1.0, 5.1, 3.8, 1.6, .2},
new double[]{1.0, 4.6, 3.2, 1.4, .2},
new double[]{1.0, 5.3, 3.7, 1.5, .2},
new double[]{1.0, 5.0, 3.3, 1.4, .2},
new double[]{2.0, 7.0, 3.2, 4.7, 1.4},
new double[]{2.0, 6.4, 3.2, 4.5, 1.5},
new double[]{2.0, 6.9, 3.1, 4.9, 1.5},
new double[]{2.0, 5.5, 2.3, 4.0, 1.3},
new double[]{2.0, 6.5, 2.8, 4.6, 1.5},
new double[]{2.0, 5.7, 2.8, 4.5, 1.3},
new double[]{2.0, 6.3, 3.3, 4.7, 1.6},
new double[]{2.0, 4.9, 2.4, 3.3, 1.0},
new double[]{2.0, 6.6, 2.9, 4.6, 1.3},
new double[]{2.0, 5.2, 2.7, 3.9, 1.4},
new double[]{2.0, 5.0, 2.0, 3.5, 1.0},
new double[]{2.0, 5.9, 3.0, 4.2, 1.5},
new double[]{2.0, 6.0, 2.2, 4.0, 1.0},
new double[]{2.0, 6.1, 2.9, 4.7, 1.4},
new double[]{2.0, 5.6, 2.9, 3.6, 1.3},
new double[]{2.0, 6.7, 3.1, 4.4, 1.4},
new double[]{2.0, 5.6, 3.0, 4.5, 1.5},
new double[]{2.0, 5.8, 2.7, 4.1, 1.0},
new double[]{2.0, 6.2, 2.2, 4.5, 1.5},
new double[]{2.0, 5.6, 2.5, 3.9, 1.1},
new double[]{2.0, 5.9, 3.2, 4.8, 1.8},
new double[]{2.0, 6.1, 2.8, 4.0, 1.3},
new double[]{2.0, 6.3, 2.5, 4.9, 1.5},
new double[]{2.0, 6.1, 2.8, 4.7, 1.2},
new double[]{2.0, 6.4, 2.9, 4.3, 1.3},
new double[]{2.0, 6.6, 3.0, 4.4, 1.4},
new double[]{2.0, 6.8, 2.8, 4.8, 1.4},
new double[]{2.0, 6.7, 3.0, 5.0, 1.7},
new double[]{2.0, 6.0, 2.9, 4.5, 1.5},
new double[]{2.0, 5.7, 2.6, 3.5, 1.0},
new double[]{2.0, 5.6, 2.4, 3.8, 1.1},
new double[]{2.0, 5.5, 2.4, 3.7, 1.0},
new double[]{2.0, 5.8, 2.7, 3.9, 1.2},
new double[]{2.0, 6.0, 2.7, 5.1, 1.6},
new double[]{2.0, 5.4, 3.0, 4.5, 1.5},
new double[]{2.0, 6.0, 3.4, 4.5, 1.6},
new double[]{2.0, 6.7, 3.1, 4.7, 1.5},
new double[]{2.0, 6.3, 2.3, 4.4, 1.3},

Data Mining

NaiveBayesClassifier • 1597
new double[] {2.0, 5.6, 3.0, 4.1, 1.3},
new double[] {2.0, 5.5, 2.5, 4.0, 1.3},
new double[] {2.0, 5.5, 2.6, 4.4, 1.2},
new double[] {2.0, 5.8, 2.6, 4.0, 1.2},
new double[] {2.0, 5.0, 2.3, 3.3, 1.0},
new double[] {2.0, 5.6, 2.7, 4.2, 1.3},
new double[] {2.0, 5.7, 3.0, 4.2, 1.2},
new double[] {2.0, 5.7, 2.9, 4.2, 1.3},
new double[] {2.0, 6.2, 2.9, 4.3, 1.3},
new double[] {2.0, 5.1, 2.5, 3.0, 1.1},
new double[] {2.0, 5.7, 2.8, 4.1, 1.3},
new double[] {3.0, 6.3, 3.3, 6.0, 2.5},
new double[] {3.0, 5.8, 2.7, 5.1, 1.9},
new double[] {3.0, 7.1, 3.0, 5.9, 2.1},
new double[] {3.0, 6.3, 2.9, 5.6, 1.8},
new double[] {3.0, 6.5, 3.0, 5.8, 2.2},
new double[] {3.0, 7.6, 3.0, 6.6, 2.1},
new double[] {3.0, 4.9, 2.5, 4.5, 1.7},
new double[] {3.0, 7.3, 2.9, 6.3, 1.8},
new double[] {3.0, 6.7, 2.5, 5.8, 1.8},
new double[] {3.0, 7.2, 3.6, 6.1, 2.5},
new double[] {3.0, 6.5, 3.2, 5.1, 2.0},
new double[] {3.0, 6.4, 2.7, 5.3, 1.9},
new double[] {3.0, 6.8, 3.0, 5.5, 2.1},
new double[] {3.0, 5.7, 2.5, 5.0, 2.0},
new double[] {3.0, 5.8, 2.8, 5.1, 2.4},
new double[] {3.0, 6.4, 3.2, 5.3, 2.3},
new double[] {3.0, 6.5, 3.0, 5.5, 1.8},
new double[] {3.0, 7.7, 3.8, 6.7, 2.2},
new double[] {3.0, 7.7, 2.6, 6.9, 2.3},
new double[] {3.0, 6.0, 2.2, 5.0, 1.5},
new double[] {3.0, 6.9, 3.2, 5.7, 2.3},
new double[] {3.0, 5.6, 2.8, 4.9, 2.0},
new double[] {3.0, 7.7, 2.8, 6.7, 2.0},
new double[] {3.0, 6.3, 2.7, 4.9, 1.8},
new double[] {3.0, 6.7, 3.3, 5.7, 2.1},
new double[] {3.0, 7.2, 3.2, 6.0, 1.8},
new double[] {3.0, 6.2, 2.8, 4.8, 1.8},
new double[] {3.0, 6.1, 3.0, 4.9, 1.8},
new double[] {3.0, 6.4, 2.8, 5.6, 2.1},
new double[] {3.0, 7.2, 3.0, 5.8, 1.6},
new double[] {3.0, 7.4, 2.8, 6.1, 1.9},
new double[] {3.0, 7.9, 3.8, 6.4, 2.0},
new double[] {3.0, 6.4, 2.8, 5.6, 2.2},
new double[] {3.0, 6.3, 2.8, 5.1, 1.5},
new double[] {3.0, 6.1, 2.6, 5.6, 1.4},
new double[] {3.0, 7.7, 3.0, 6.1, 2.3},
new double[] {3.0, 6.3, 3.4, 5.6, 2.4},
new double[] {3.0, 6.4, 3.1, 5.5, 1.8},
new double[] {3.0, 6.0, 3.0, 4.8, 1.8},
new double[] {3.0, 6.9, 3.1, 5.4, 2.1},
new double[] {3.0, 6.7, 3.1, 5.6, 2.4},
new double[] {3.0, 6.9, 3.1, 5.1, 2.3},
new double[] {3.0, 5.8, 2.7, 5.1, 1.9},
new double[] {3.0, 6.8, 3.2, 5.9, 2.3},
new double[]{3.0, 6.7, 3.3, 5.7, 2.5},
new double[]{3.0, 6.7, 3.0, 5.2, 2.3},
new double[]{3.0, 6.3, 2.5, 5.0, 1.9},
new double[]{3.0, 6.5, 3.0, 5.2, 2.0},
new double[]{3.0, 6.2, 3.4, 5.4, 2.3},
new double[]{3.0, 5.9, 3.0, 5.1, 1.8});

public static void Main(String[] args)
{

    /* Data corrections described in the KDD data mining archive */
    irisFisherData[34][4] = 0.1;
    irisFisherData[37][2] = 3.1;
    irisFisherData[37][3] = 1.5;

    /* Train first 140 patterns of the iris Fisher Data */
    int[] irisClassificationData =
        new int[irisFisherData.Length - 10];
    double[][] irisContinuousData =
        new double[irisFisherData.Length - 10][];
    for (int i = 0; i < irisFisherData.Length - 10; i++)
    {
        irisContinuousData[i] =
            new double[irisFisherData[0].Length - 1];
    }
    for (int i = 0; i < irisFisherData.Length - 10; i++)
    {
        irisClassificationData[i] = (int)irisFisherData[i][0] - 1;
        Array.Copy(irisFisherData[i], 1, irisContinuousData[i], 0,
            irisFisherData[0].Length - 1);
    }

    int nNominal = 0; /* no nominal input attributes */
    int nContinuous = 4; /* four continuous input attributes */
    int nClasses = 3; /* three classification categories */

    NaiveBayesClassifier nbTrainer =
        new NaiveBayesClassifier(nContinuous, nNominal, nClasses);
    for (int i = 0; i < nContinuous; i++)
        nbTrainer.CreateContinuousAttribute(
            new NormalDistribution());
    nbTrainer.Train(irisContinuousData, null,
        irisClassificationData);

    int[][] classErrors = nbTrainer.GetTrainingErrors();

    Console.Out.WriteLine(
        " Iris Classification Training Error Rates");
    Console.Out.WriteLine(
        "-----------------------------------------------");
    Console.Out.WriteLine(
        " Setosa Versicolour Virginica | Total");
    Console.Out.WriteLine(" ", classErrors[0][0] + " /" +

classErrors[0][1] + " " + classErrors[1][0] + "/" +
classErrors[1][1] + " " + classErrors[2][0] + "/" +
classErrors[2][1] + " " + classErrors[3][0] + 
"/" + classErrors[3][1]);
Console.Out.WriteLine("------------------------------------------------


");// Classify last 10 iris data patterns
/* with the trained classifier
*/
double[] continuousInput =
new double[(irisFisherData[0].Length - 1)];
double[] classifiedProbabilities = new double[nClasses];

Console.Out.WriteLine("Probabilities for Incorrect Classifications");
Console.Out.WriteLine(" Predicted ");
Console.Out.WriteLine(" Class | Class | P(0) P(1) P(2) ");
Console.Out.WriteLine("-------------------------------------------------------");
for (int i = 0; i < 10; i++)
{
    int targetClassification = (int)irisFisherData[(irisFisherData.Length - 10) + i][0] - 1;
    Array.Copy(irisFisherData[(irisFisherData.Length - 10) + i],
                1, continuousInput, 0, (irisFisherData[0].Length - 1));
    classifiedProbabilities = nbTrainer.Probabilities(continuousInput, null);
    int classification = nbTrainer.PredictClass(continuousInput, null);
    if (classification == 0)
        Console.Out.Write("Setosa | ");
    else if (classification == 1)
        Console.Out.Write("Versicolour | ");
    else if (classification == 2)
        Console.Out.Write("Virginica | ");
    else
        Console.Out.Write("Missing | ");
    if (targetClassification == 0)
        Console.Out.Write(" Setosa | ");
    else if (targetClassification == 1)
        Console.Out.Write(" Versicolour | ");
    else if (targetClassification == 2)
        Console.Out.Write(" Virginica | ");
    else
        Console.Out.Write(" Missing | ");
    for (int j = 0; j < nClasses; j++)
    {
        Object[] pArgs = new Object[] {
            (double)classifiedProbabilities[j]};
        Console.Out.Write(" {0, 2:f3} ", pArgs);
    }
}
The Naive Bayes classifier incorrectly classifies 6 of the 150 training patterns.

Iris Classification Training Error Rates
-------------------------------------------------
<table>
<thead>
<tr>
<th>Setosa</th>
<th>Versicolour</th>
<th>Virginica</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/50</td>
<td>0/50</td>
<td>20/40</td>
<td>20/140</td>
</tr>
</tbody>
</table>
-------------------------------------------------

Probabilities for Incorrect Classifications

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>Virginica</th>
<th>P(0)</th>
<th>P(1)</th>
<th>P(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.436</td>
<td>0.564</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.466</td>
<td>0.534</td>
</tr>
<tr>
<td>Versicolour</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.542</td>
<td>0.458</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.441</td>
<td>0.559</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.412</td>
<td>0.588</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.466</td>
<td>0.534</td>
</tr>
<tr>
<td>Versicolour</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.542</td>
<td>0.458</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.515</td>
<td>0.485</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.460</td>
<td>0.540</td>
</tr>
<tr>
<td>Versicolour</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.551</td>
<td>0.449</td>
</tr>
</tbody>
</table>

Example 2: Nominal Attribute Usage

This example trains a Naive Bayes classifier using 24 training patterns with four nominal input attributes.

The first nominal attribute has three classifications and the others have two. The target classifications are contact lens prescriptions: hard, soft or neither recommended. These data are benchmark data from the Knowledge Discovery Databases archive maintained at the University of California, Irvine:

```csharp
using System;
using Imsl.DataMining;

public class NaiveBayesClassifierEx2
{
    public static void Main(String[] args)
    {
        int[][] contactLensData = new int[][]{
            new int[]{1, 1, 1, 1}, new int[]{1, 1, 1, 2},
            new int[]{1, 1, 2, 1}, new int[]{1, 1, 2, 2},
            new int[]{1, 2, 1, 1}, new int[]{1, 2, 1, 2},
            new int[]{1, 2, 2, 1}, new int[]{1, 2, 2, 2},
```
new int[]{2, 1, 1, 1}, new int[]{2, 1, 2, 1},
new int[]{2, 2, 1, 1}, new int[]{2, 2, 1, 2},
new int[]{3, 1, 1, 1}, new int[]{3, 1, 1, 2},
new int[]{3, 1, 2, 1}, new int[]{3, 1, 2, 2},
new int[]{3, 2, 1, 1}, new int[]{3, 2, 1, 2},
new int[]{3, 2, 2, 1}, new int[]{3, 2, 2, 2};

int[] classificationData = new int[]{
    3, 2, 3, 1, 3, 2, 3, 1, 3, 2, 3, 1,
    3, 2, 3, 3, 3, 3, 3, 1, 3, 2, 3, 3
};
int nNominal = 4;
int[] categories = new int[]{3, 2, 2, 2};
for (int i = 0; i < nNominal; i++)
    nbTrainer.CreateNominalAttribute(categories[i]);

int[][] classErrors = nbTrainer.GetTrainingErrors();

/* Classify all patterns with the trained classifier */
int[] nominalInput = new int[contactLensData[0].Length];
double[] classifiedProbabilities = new double[3];

*/
null, nominalInput);
    int classification =
        nbTrainer.PredictClass(null, nominalInput);
    double error = nbTrainer.ClassError(null, nominalInput,
        classificationData[i]);
    if (classification == 0)
        Console.Out.Write(" Hard | ");
    else if (classification == 1)
        Console.Out.Write(" Soft | ");
    else if (classification == 2)
        Console.Out.Write(" Neither | ");
    else
        Console.Out.Write(" Missing | ");
    if (classificationData[i] == 0)
        Console.Out.Write(" Hard | ");
    else if (classificationData[i] == 1)
        Console.Out.Write(" Soft | ");
    else if (classificationData[i] == 2)
        Console.Out.Write(" Neither | ");
    else
        Console.Out.Write(" Missing | ");
    for (int j = 0; j < 3; j++)
    {
        Object[] pArgs = new Object[] { (double)classifiedProbabilities[j] };
        Console.Out.Write(" {0, 2:f3} ", pArgs);
    }
    Console.Out.WriteLine(" | " + error);
}]

Output

Contact Lense Error Rates
---------------------------------------
Hard  Soft  Neither | Total
0/4   0/5   1/15    | 1/24
---------------------------------------

Probabilities for Incorrect Classifications

Predicted

Data Mining NaiveBayesClassifier • 1603
<table>
<thead>
<tr>
<th>Class</th>
<th>Class</th>
<th>P(0)</th>
<th>P(1)</th>
<th>P(2)</th>
<th>classification error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.044</td>
<td>0.130</td>
<td>0.827</td>
<td>0.17328273572243</td>
</tr>
<tr>
<td>Soft</td>
<td>Soft</td>
<td>0.174</td>
<td>0.622</td>
<td>0.203</td>
<td>0.377703751596285</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.186</td>
<td>0.018</td>
<td>0.795</td>
<td>0.204814844531185</td>
</tr>
<tr>
<td>Hard</td>
<td>Hard</td>
<td>0.724</td>
<td>0.086</td>
<td>0.190</td>
<td>0.27622138817044</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.019</td>
<td>0.154</td>
<td>0.827</td>
<td>0.173119280942829</td>
</tr>
<tr>
<td>Soft</td>
<td>Soft</td>
<td>0.076</td>
<td>0.724</td>
<td>0.200</td>
<td>0.275800758853599</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.092</td>
<td>0.024</td>
<td>0.884</td>
<td>0.116366478280669</td>
</tr>
<tr>
<td>Hard</td>
<td>Hard</td>
<td>0.524</td>
<td>0.166</td>
<td>0.310</td>
<td>0.475967127191525</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.025</td>
<td>0.113</td>
<td>0.862</td>
<td>0.137948891733597</td>
</tr>
<tr>
<td>Soft</td>
<td>Soft</td>
<td>0.118</td>
<td>0.633</td>
<td>0.248</td>
<td>0.366677564499419</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.113</td>
<td>0.017</td>
<td>0.870</td>
<td>0.1300941614979</td>
</tr>
<tr>
<td>Hard</td>
<td>Hard</td>
<td>0.606</td>
<td>0.108</td>
<td>0.286</td>
<td>0.394395320215918</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.011</td>
<td>0.133</td>
<td>0.856</td>
<td>0.143807127534487</td>
</tr>
<tr>
<td>Soft</td>
<td>Soft</td>
<td>0.050</td>
<td>0.714</td>
<td>0.236</td>
<td>0.286218909502689</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.054</td>
<td>0.021</td>
<td>0.925</td>
<td>0.0747706562988114</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.394</td>
<td>0.187</td>
<td>0.419</td>
<td>0.58120710821219</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.023</td>
<td>0.068</td>
<td>0.909</td>
<td>0.090752977094107</td>
</tr>
<tr>
<td>Soft</td>
<td>Soft</td>
<td>0.142</td>
<td>0.509</td>
<td>0.349</td>
<td>0.650925868235811</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.099</td>
<td>0.010</td>
<td>0.891</td>
<td>0.109251850144518</td>
</tr>
<tr>
<td>Hard</td>
<td>Hard</td>
<td>0.599</td>
<td>0.071</td>
<td>0.330</td>
<td>0.401383015922963</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.010</td>
<td>0.081</td>
<td>0.909</td>
<td>0.090658834477251</td>
</tr>
<tr>
<td>Soft</td>
<td>Soft</td>
<td>0.062</td>
<td>0.594</td>
<td>0.344</td>
<td>0.406256189123439</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.047</td>
<td>0.012</td>
<td>0.941</td>
<td>0.0590094638695026</td>
</tr>
<tr>
<td>Neither</td>
<td>Neither</td>
<td>0.391</td>
<td>0.124</td>
<td>0.485</td>
<td>0.514955474332508</td>
</tr>
</tbody>
</table>

### Example 3: Naive Bayes Classifier Using User Supplied Probability Function

This example is the same as Example 1, using Fisher’s (1936) Iris data to train a Naive Bayes classifier using 140 of the 150 continuous patterns, then classifies ten unknown plants using their sepal and petal measurements.

Instead of using the NormalDistribution class from the Imsl.Stat namespace, a user supplied normal (Gaussian) distribution is used. Rather than calculating the means and standard deviations from the data, as is done by the NormalDistribution’s Eval(double[]) method, the user supplied class requires the means and standard deviations in the class constructor. The output is the same as in Example 1, since the means and standard deviations in this example are simply rounded means and standard deviations of the actual data subset by target classifications.

```csharp
using System;
using Imsl.DataMining;
using IProbabilityDistribution = Imsl.Stat.IProbabilityDistribution;

public class NaiveBayesClassifierEx3
{
    private static double[][] irisFisherData = new double[][]{
        new double[]{1.0, 5.1, 3.5, 1.4, .2},
        new double[]{1.0, 4.9, 3.0, 1.4, .2},
        new double[]{1.0, 4.7, 3.2, 1.3, .2},
        new double[]{1.0, 4.6, 3.1, 1.5, .2},
    };
```
new double[] {1.0, 5.0, 3.6, 1.4, .2},
new double[] {1.0, 5.4, 3.9, 1.7, .4},
new double[] {1.0, 4.6, 3.4, 1.4, .3},
new double[] {1.0, 5.0, 3.4, 1.5, .2},
new double[] {1.0, 4.4, 2.9, 1.4, .2},
new double[] {1.0, 4.9, 3.1, 1.5, .1},
new double[] {1.0, 5.4, 3.7, 1.5, .2},
new double[] {1.0, 4.8, 3.4, 1.6, .2},
new double[] {1.0, 4.8, 3.0, 1.4, .1},
new double[] {1.0, 4.3, 3.0, 1.1, .1},
new double[] {1.0, 5.8, 4.0, 1.2, .2},
new double[] {1.0, 5.7, 4.4, 1.5, .4},
new double[] {1.0, 5.4, 3.9, 1.3, .4},
new double[] {1.0, 5.1, 3.5, 1.4, .3},
new double[] {1.0, 5.7, 3.8, 1.7, .3},
new double[] {1.0, 5.1, 3.8, 1.5, .3},
new double[] {1.0, 5.4, 3.4, 1.7, .2},
new double[] {1.0, 5.1, 3.7, 1.5, .4},
new double[] {1.0, 4.6, 3.6, 1.0, .2},
new double[] {1.0, 5.1, 3.3, 1.7, .5},
new double[] {1.0, 4.8, 3.4, 1.9, .2},
new double[] {1.0, 5.0, 3.0, 1.6, .2},
new double[] {1.0, 5.0, 3.4, 1.6, .4},
new double[] {1.0, 5.2, 3.5, 1.5, .2},
new double[] {1.0, 5.2, 3.4, 1.4, .2},
new double[] {1.0, 4.7, 3.2, 1.6, .2},
new double[] {1.0, 4.8, 3.1, 1.6, .2},
new double[] {1.0, 5.4, 3.4, 1.5, .4},
new double[] {1.0, 5.2, 4.1, 1.5, .1},
new double[] {1.0, 5.5, 4.2, 1.4, .2},
new double[] {1.0, 4.9, 3.1, 1.5, .2},
new double[] {1.0, 5.0, 3.2, 1.2, .2},
new double[] {1.0, 5.5, 3.5, 1.3, .2},
new double[] {1.0, 4.9, 3.6, 1.4, .1},
new double[] {1.0, 4.4, 3.0, 1.3, .2},
new double[] {1.0, 5.1, 3.4, 1.5, .2},
new double[] {1.0, 5.0, 3.5, 1.3, .3},
new double[] {1.0, 4.5, 2.3, 1.3, .3},
new double[] {1.0, 4.4, 3.2, 1.3, .2},
new double[] {1.0, 5.0, 3.5, 1.6, .6},
new double[] {1.0, 5.1, 3.8, 1.9, .4},
new double[] {1.0, 4.8, 3.0, 1.4, .3},
new double[] {1.0, 5.1, 3.8, 1.6, .2},
new double[] {1.0, 4.6, 3.2, 1.4, .2},
new double[] {1.0, 5.3, 3.7, 1.5, .2},
new double[] {1.0, 5.0, 3.3, 1.4, .2},
new double[] {2.0, 7.0, 3.2, 4.7, 1.4},
new double[] {2.0, 6.4, 3.2, 4.5, 1.5},
new double[] {2.0, 6.9, 3.1, 4.9, 1.8},
new double[] {2.0, 5.5, 2.3, 4.0, 1.3},
new double[] {2.0, 6.5, 2.8, 4.6, 1.5},
new double[] {2.0, 5.7, 2.8, 4.5, 1.3},
new double[] {2.0, 6.3, 3.3, 4.7, 1.6},
new double[] {2.0, 4.9, 2.4, 3.3, 1.0},
new double[] {2.0, 6.6, 2.9, 4.6, 1.3},
new double[] {2.0, 5.2, 2.7, 3.9, 1.4},
new double[] {2.0, 5.0, 2.0, 3.5, 1.0},
new double[] {2.0, 5.9, 3.0, 4.2, 1.5},
new double[] {2.0, 6.0, 2.2, 4.0, 1.0},
new double[] {2.0, 6.1, 2.9, 4.7, 1.4},
new double[] {2.0, 5.6, 2.9, 3.6, 1.3},
new double[] {2.0, 6.7, 3.1, 4.4, 1.4},
new double[] {2.0, 5.6, 3.0, 4.5, 1.5},
new double[] {2.0, 5.8, 2.7, 4.1, 1.0},
new double[] {2.0, 6.2, 2.2, 4.5, 1.5},
new double[] {2.0, 5.6, 2.5, 3.9, 1.1},
new double[] {2.0, 5.9, 3.2, 4.8, 1.8},
new double[] {2.0, 6.1, 2.8, 4.0, 1.3},
new double[] {2.0, 6.3, 2.5, 4.9, 1.5},
new double[] {2.0, 6.1, 2.8, 4.7, 1.2},
new double[] {2.0, 6.4, 2.9, 4.3, 1.3},
new double[] {2.0, 6.6, 3.0, 4.4, 1.4},
new double[] {2.0, 6.8, 2.8, 4.8, 1.4},
new double[] {2.0, 6.7, 3.0, 5.0, 1.7},
new double[] {2.0, 6.0, 2.9, 4.5, 1.5},
new double[] {2.0, 5.7, 2.6, 3.5, 1.0},
new double[] {2.0, 5.5, 2.4, 3.8, 1.1},
new double[] {2.0, 5.5, 2.4, 3.7, 1.0},
new double[] {2.0, 5.8, 2.7, 3.9, 1.2},
new double[] {2.0, 6.0, 2.7, 5.1, 1.6},
new double[] {2.0, 5.4, 3.0, 4.5, 1.5},
new double[] {2.0, 6.0, 3.4, 4.5, 1.6},
new double[] {2.0, 6.7, 3.1, 4.7, 1.5},
new double[] {2.0, 6.3, 2.3, 4.4, 1.3},
new double[] {2.0, 5.6, 3.0, 4.1, 1.3},
new double[] {2.0, 5.5, 2.5, 4.0, 1.3},
new double[] {2.0, 5.5, 2.6, 4.4, 1.2},
new double[] {2.0, 6.1, 3.0, 4.6, 1.4},
new double[] {2.0, 5.8, 2.6, 4.0, 1.2},
new double[] {2.0, 5.0, 2.3, 3.3, 1.0},
new double[] {2.0, 5.6, 2.7, 4.2, 1.3},
new double[] {2.0, 5.7, 3.0, 4.2, 1.2},
new double[] {2.0, 5.7, 2.9, 4.2, 1.3},
new double[] {2.0, 6.2, 2.9, 4.3, 1.3},
new double[] {2.0, 5.1, 2.5, 3.0, 1.1},
new double[] {2.0, 5.7, 2.8, 4.1, 1.3},
new double[] {3.0, 6.3, 3.3, 6.0, 2.5},
new double[] {3.0, 5.8, 2.7, 5.1, 1.9},
new double[] {3.0, 7.1, 3.0, 5.9, 2.1},
new double[] {3.0, 6.3, 2.9, 5.6, 1.8},
new double[] {3.0, 6.5, 3.0, 5.8, 2.2},
new double[] {3.0, 7.6, 3.0, 6.6, 2.1},
new double[] {3.0, 4.9, 2.5, 4.5, 1.7},
new double[] {3.0, 7.3, 2.9, 6.3, 1.8},
new double[] {3.0, 6.7, 2.5, 5.8, 1.8},
new double[] {3.0, 7.2, 3.6, 6.1, 2.5},
new double[] {3.0, 6.5, 3.2, 5.1, 2.0},
new double[] {3.0, 6.4, 2.7, 5.3, 1.9},
new double[] {3.0, 6.8, 3.0, 5.5, 2.1},
new double[] {3.0, 5.7, 2.5, 5.0, 2.0},
new double[] {3.0, 5.8, 2.8, 5.1, 2.4},
new double[] {3.0, 6.4, 3.2, 5.3, 2.3},
new double[] {3.0, 6.5, 3.0, 5.5, 1.8},
new double[] {3.0, 7.7, 3.8, 6.7, 2.2},
new double[] {3.0, 7.7, 2.6, 6.9, 2.3},
new double[] {3.0, 6.0, 2.2, 5.0, 1.5},
new double[] {3.0, 6.9, 3.2, 5.7, 2.3},
new double[] {3.0, 5.6, 2.8, 4.9, 2.0},
new double[] {3.0, 7.7, 2.8, 6.7, 2.0},
new double[] {3.0, 6.3, 2.7, 4.9, 1.8},
new double[] {3.0, 6.7, 3.3, 5.7, 2.1},
new double[] {3.0, 7.2, 3.2, 6.0, 1.8},
new double[] {3.0, 6.2, 2.8, 4.8, 1.8},
new double[] {3.0, 6.1, 3.0, 4.9, 1.8},
new double[] {3.0, 6.4, 2.8, 5.6, 2.1},
new double[] {3.0, 7.2, 3.0, 5.8, 1.6},
new double[] {3.0, 7.4, 2.8, 6.1, 1.9},
new double[] {3.0, 7.9, 3.8, 6.4, 2.0},
new double[] {3.0, 6.4, 2.8, 5.6, 2.2},
new double[] {3.0, 6.3, 2.8, 5.1, 1.5},
new double[] {3.0, 6.1, 2.6, 5.6, 1.4},
new double[] {3.0, 7.7, 3.0, 6.1, 2.3},
new double[] {3.0, 6.3, 3.4, 5.6, 2.4},
new double[] {3.0, 6.4, 3.1, 5.5, 1.8},
new double[] {3.0, 6.0, 3.0, 4.8, 1.8},
new double[] {3.0, 6.9, 3.1, 5.4, 2.1},
new double[] {3.0, 6.7, 3.1, 5.6, 2.4},
new double[] {3.0, 6.9, 3.1, 5.1, 2.3},
new double[] {3.0, 5.8, 2.7, 5.1, 1.9},
new double[] {3.0, 6.8, 3.2, 5.9, 2.3},
new double[] {3.0, 6.7, 3.3, 5.7, 2.5},
new double[] {3.0, 6.7, 3.0, 5.2, 2.3},
new double[] {3.0, 6.3, 2.5, 5.0, 1.9},
new double[] {3.0, 6.5, 3.0, 5.2, 2.0},
new double[] {3.0, 6.2, 3.4, 5.4, 2.3},
new double[] {3.0, 5.9, 3.0, 5.1, 1.8};

public static void Main(System.String[] args)
{
    /* Data corrections described in the KDD data mining archive */
    irisFisherData[34][4] = 0.1;
    irisFisherData[37][2] = 3.1;
    irisFisherData[37][3] = 1.5;

    /* Train first 140 patterns of the iris Fisher Data */
    int[] irisClassificationData =
        new int[irisFisherData.Length - 10];
    double[][] irisContinuousData =
        new double[irisFisherData.Length - 10][];
    for (int i = 0; i < irisFisherData.Length - 10; i++)
    {
        irisContinuousData[i] =
            new double[irisFisherData[0].Length - 1];
    }
    for (int i = 0; i < irisFisherData.Length - 10; i++)
    {

irisClassificationData[i] = (int) irisFisherData[i][0] - 1;
Array.Copy(irisFisherData[i], 1, irisContinuousData[i], 0, irisFisherData[0].Length - 1);

int nNominal = 0; /* no nominal input attributes */
int nContinuous = 4; /* four continuous input attributes */
int nClasses = 3; /* three classification categories */

NaiveBayesClassifier nbTrainer =
    new NaiveBayesClassifier(nContinuous, nNominal, nClasses);

double[][] means = new double[][nClasses]{
    new double[]{5.06, 5.94, 6.58},
    new double[]{3.42, 2.8, 2.97},
    new double[]{1.5, 4.3, 5.6},
    new double[]{0.25, 1.33, 2.1}
};
double[][] stddev = new double[][nClasses]{
    new double[]{0.35, 0.52, 0.64},
    new double[]{0.38, 0.3, 0.32},
    new double[]{0.17, 0.47, 0.55},
    new double[]{0.12, 0.198, 0.275}
};

for (int i = 0; i < nContinuous; i++)
{
    IProbabilityDistribution[] pdf =
        new IProbabilityDistribution[nClasses];
    for (int j = 0; j < nClasses; j++)
    {
        pdf[j] = new TestGaussFcn1(means[i][j], stddev[i][j]);
    }
    nbTrainer.CreateContinuousAttribute(pdf);
}

nbTrainer.Train(irisContinuousData, null, irisClassificationData);

int[][] classErrors = nbTrainer.GetTrainingErrors();

System.Console.Out.WriteLine(" Iris Classification Error Rates");
System.Console.Out.WriteLine("------------------------------------------------");
System.Console.Out.WriteLine(" Setosa Versicolour Virginica | Total");
System.Console.Out.WriteLine("------------------------------------------------


/

 NaiveBayesClassifier

IMSL C# Numerical Library
* with the trained classifier
 */
 double[] continuousInput =
    new double[(irisFisherData[0].Length - 1)];
 double[] classifiedProbabilities = new double[nClasses];

 System.Console.Out.WriteLine(" Predicted ");
 System.Console.Out.WriteLine(" Class | Class | P(0) P(1) P(2) ");
 System.Console.Out.WriteLine("-------------------------------------------------------");
 for (int i = 0; i < 10; i++)
 {
    int targetClassification = (int)
       irisFisherData[(irisFisherData.Length - 10) + i][0] - 1;
    Array.Copy(irisFisherData[(irisFisherData.Length - 10) + i],
       1, continuousInput, 0, (irisFisherData[0].Length - 1));
    classifiedProbabilities =
       nbTrainer.Probabilities(continuousInput, null);
    int classification =
       nbTrainer.PredictClass(continuousInput, null);
    if (classification == 0)
    {
       System.Console.Out.Write("Setosa | ");
    } else if (classification == 1)
    {
       System.Console.Out.Write("Versicolour | ");
    } else if (classification == 2)
    {
       System.Console.Out.Write("Virginica | ");
    } else
    {
       System.Console.Out.Write("Missing | ");
    }
    if (targetClassification == 0)
    {
       System.Console.Out.Write(" Setosa | ");
    } else if (targetClassification == 1)
    {
       System.Console.Out.Write(" Versicolour | ");
    } else if (targetClassification == 2)
    {
       System.Console.Out.Write(" Virginica | ");
    } else
    {
       System.Console.Out.Write(" Missing | ");
    }
for (int j = 0; j < nClasses; j++)
{
    System.Object[] pArgs = new System.Object[] {
        (double)classifiedProbabilities[j] 
    };
    System.Console.Out.Write(" {0, 2:f3} ", pArgs);
}
System.Console.Out.WriteLine("\n");

public class TestGaussFcn1 : IProbabilityDistribution
{
    virtual public System.Object[] GetParameters()
    {
        System.Object[] parms = new System.Object[2];
        parms[0] = this.mean;
        parms[1] = this.stdev;
        return parms;
    }
    private double mean;
    private double stdev;
    public TestGaussFcn1(double mean, double stdev)
    {
        this.mean = mean;
        this.stdev = stdev;
    }
    public virtual double[] Eval(double[] xData)
    {
        double[] pdf = new double[xData.Length];
        for (int i = 0; i < xData.Length; i++)
        {
            pdf[i] = Eval(xData[i], null);
        }
        return pdf;
    }
    public virtual double[] Eval(double[] xData,
        System.Object[] Params)
    {
        double[] pdf = new double[xData.Length];
        for (int i = 0; i < xData.Length; i++)
        {
            pdf[i] = Eval(xData[i], Params);
        }
        return pdf;
    }
    public virtual double Eval(double xData,
        System.Object[] Params)
    {
        return GaussianPdf(xData, mean, stdev);
    }
    private double GaussianPdf(double x, double mean, double stdev)
double e, phi2, z, s;
double sqrt_pi2 = 2.506628274631; /* sqrt(2*pi) */
if (System.Double.IsNaN(x))
{
    return System.Double.NaN;
}
if (System.Double.IsNaN(mean) || System.Double.IsNaN(stdev))
{
    return System.Double.NaN;
}
else
{
    z = x;
    z -= mean;
    s = stdev;
    phi2 = sqrt_pi2 * s;
    e = (- 0.5) * (z * z) / (s * s);
    return System.Math.Exp(e) / phi2;
}

Output
The Naive Bayes classifier incorrectly classifies 6 of the 150 training patterns.

Iris Classification Error Rates
---------------------------------
<table>
<thead>
<tr>
<th>Setosa</th>
<th>Versicolour</th>
<th>Virginica</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/50</td>
<td>2/50</td>
<td>4/40</td>
<td>6/140</td>
</tr>
</tbody>
</table>
---------------------------------

Probabilities for Incorrect Classifications
Predicted
-------------------------------------
<table>
<thead>
<tr>
<th>Class</th>
<th>Class</th>
<th>P(0)</th>
<th>P(1)</th>
<th>P(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.051</td>
<td>0.949</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.048</td>
<td>0.952</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.001</td>
<td>0.999</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Virginica</td>
<td>Virginica</td>
<td>0.000</td>
<td>0.126</td>
<td>0.874</td>
</tr>
</tbody>
</table>
Chapter 27: Neural Nets

Types

\begin{itemize}
  \item \texttt{class FeedForwardNetwork} .................................................. 1639
  \item \texttt{class Layer} ................................................................. 1654
  \item \texttt{class InputLayer} ......................................................... 1655
  \item \texttt{class HiddenLayer} ....................................................... 1656
  \item \texttt{class OutputLayer} ....................................................... 1657
  \item \texttt{class Node} ............................................................... 1659
  \item \texttt{class InputNode} .......................................................... 1660
  \item \texttt{class Perceptron} ......................................................... 1661
  \item \texttt{class OutputPerceptron} ............................................... 1662
  \item \texttt{class IActivation} ....................................................... 1663
  \item \texttt{structure Activation} ................................................. 1664
  \item \texttt{class Link} ............................................................... 1665
  \item \texttt{class ITrainer} .......................................................... 1666
  \item \texttt{class QuasiNewtonTrainer} ........................................... 1668
  \item \texttt{interface QuasiNewtonTrainer.IError} ............................ 1674
  \item \texttt{class LeastSquaresTrainer} ........................................... 1675
  \item \texttt{class EpochTrainer} .................................................... 1680
  \item \texttt{class BinaryClassification} ....................................... 1683
  \item \texttt{class MultiClassification} ......................................... 1727
  \item \texttt{class ScaleFilter} ...................................................... 1742
    \item \texttt{enumeration ScaleFilter.ScalingMethod} ....................... 1751
  \item \texttt{class UnsupervisedNominalFilter} .................................. 1752
  \item \texttt{class UnsupervisedOrdinalFilter} .................................. 1756
    \item \texttt{enumeration UnsupervisedOrdinalFilter.TransformMethod} . 1761
  \item \texttt{class TimeSeriesFilter} .............................................. 1762
  \item \texttt{class TimeSeriesClassFilter} ...................................... 1764
  \item \texttt{class Network} .......................................................... 1786
\end{itemize}
Usage Notes

Neural Networks - An Overview

Today, neural networks are used to solve a wide variety of problems, some of which have been solved by existing statistical methods, and some of which have not. These applications fall into one of the following three categories:

- **Forecasting**: predicting one or more quantitative outcomes from both quantitative and categorical input data,
- **Classification**: classifying input data into one of two or more categories, or
- **Statistical pattern recognition**: uncovering patterns, typically spatial or temporal, among a set of variables.

Forecasting, pattern recognition and classification problems are not new. They existed years before the discovery of neural network solutions in the 1980’s. What is new is that neural networks provide a single framework for solving so many traditional problems and, in some cases, extend the range of problems that can be solved.

Traditionally, these problems have been solved using a variety of well known statistical methods:

- linear regression and general least squares,
- logistic regression and discrimination,
- principal component analysis,
- discriminant analysis,
- $k$-nearest neighbor classification, and
- ARMA and non-linear ARMA time series forecasts.

In many cases, simple neural network configurations yield the same solution as many traditional statistical applications. For example, a single-layer, feed-forward neural network with linear activation for its output perceptron is equivalent to a general linear regression fit. Neural networks can provide more accurate and robust solutions for problems where traditional methods do not completely apply.

Mandic and Chambers (2001) point out that traditional methods for time series forecasting are unsuitable when a time series:

- is non-stationary,
- has large amounts of noise, such as a biomedical series, or
- is too short.
ARIMA and other traditional time series approaches can produce poor forecasts when one or more of the above conditions exist. The forecasts of ARMA and non-linear ARMA (NARMA) depend heavily upon key assumptions about the model or underlying relationship between the output of the series and its patterns.

Neural networks, on the other hand, adapt to changes in a non-stationary series and can produce reliable forecasts even when the series contains a good deal of noise or when only a short series is available for training. Neural networks provide a single tool for solving many problems traditionally solved using a wide variety of statistical tools and for solving problems when traditional methods fail to provide an acceptable solution.

Although neural network solutions to forecasting, pattern recognition, and classification problems can be very different, they are always the result of computations that proceed from the network inputs to the network outputs. The network inputs are referred to as *patterns*, and outputs are referred to as *classes*. Frequently the flow of these computations is in one direction, from the network input patterns to its outputs. Networks with forward-only flow are referred to as feed-forward networks.

![A 2-layer, Feed-Forward Network with 4 Inputs and 2 Outputs](Figure 1)

Other networks, such as recurrent neural networks, allow data and information to flow in both directions,
A neural network is defined not only by its architecture and flow, or interconnections, but also by computations used to transmit information from one node or input to another node. These computations are determined by network weights. The process of fitting a network to existing data to determine these weights is referred to as training the network, and the data used in this process are referred to as patterns. Individual network inputs are referred to as attributes and outputs are referred to as classes. Many terms used to describe neural networks are synonymous to common statistical terminology.

Table 1. Synonyms between Neural Network and Common Statistical Terminology
Neural Network Terminology

<table>
<thead>
<tr>
<th>Neural Network Terminology</th>
<th>Traditional Statistical Terminology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>Model Fitting</td>
<td>Estimating unknown parameters or coefficients in the analysis.</td>
</tr>
<tr>
<td>Patterns</td>
<td>Cases or Observations</td>
<td>A single observation of all input and output variables.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Independent variables</td>
<td>Inputs to the network or model.</td>
</tr>
<tr>
<td>Classes</td>
<td>Dependent variables</td>
<td>Outputs from the network or model calculations.</td>
</tr>
</tbody>
</table>

Neural Networks – History and Terminology

The Threshold Neuron

McCulloch and Pitts (1943) wrote one of the first published works on neural networks. In their paper, they describe the threshold neuron as a model for how the human brain stores and processes information.

![Figure 3. The McCulloch and Pitts Threshold Neuron](image)

All inputs to a threshold neuron are combined into a single number, $Z$, using the following weighted sum:

$$Z = \sum_{i=1}^{m} w_i x_i - \mu$$

where $w_i$ is the weight associated with the $i$-th input (attribute) $x_i$. The term $\mu$ in this calculation is referred to as the bias term. In traditional statistical terminology, it might be referred to as the intercept. The weights and bias terms in this calculation are estimated during network training.
In McCulloch and Pitt's description of the threshold neuron, the neuron does not respond to its inputs unless $Z$ is greater than zero. If $Z$ is greater than zero then the output from this neuron is set to 1. If $Z$ is less than zero the output is zero:

$$Y = \begin{cases} 
1 & \text{if } Z > 0 \\
0 & \text{if } Z \leq 0 
\end{cases}$$

where $Y$ is the neuron’s output.

For years following their 1943 paper, interest in the McCulloch and Pitts neural network was limited to theoretical discussions, such as those of Hebb (1949), about learning, memory, and the brain’s structure.

**The Perceptron**

The McCulloch and Pitts neuron is also referred to as a threshold neuron since it abruptly changes its output from 0 to 1 when its potential, $Z$, crosses a threshold. Mathematically, this behavior can be viewed as a step function that maps the neuron’s potential, $Z$, to the neuron’s output, $Y$.

Rosenblatt (1958) extended the McCulloch and Pitts threshold neuron by replacing this step function with a continuous function that maps $Z$ to $Y$. The Rosenblatt neuron is referred to as the perceptron, and the continuous function mapping $Z$ to $Y$ makes it easier to train a network of perceptrons than a network of threshold neurons.

Unlike the threshold neuron, the perceptron produces analog output rather than the threshold neuron’s purely binary output. Carefully selecting the analog function makes Rosenblatt’s perceptron differentiable, whereas the threshold neuron is not. This simplifies the training algorithm.

Like the threshold neuron, Rosenblatt’s perceptron starts by calculating a weighted sum of its inputs, $Z = \sum_{i=1}^{m} w_ix_i - \mu$. This is referred to as the perceptron’s potential.

Rosenblatt’s perceptron calculates its analog output from its potential. There are many choices for this calculation. The function used for this calculation is referred to as the activation function in Figure 4 below.
As shown in Figure 4, perceptrons consist of the following five components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>$X_1, X_2, X_3$</td>
</tr>
<tr>
<td>Input Weights</td>
<td>$W_1, W_2, W_3$</td>
</tr>
<tr>
<td>Potential</td>
<td>$Z = \sum_{i=1}^{3} W_i X_i - \mu$, where $\mu$ is a bias correction.</td>
</tr>
<tr>
<td>Activation Function</td>
<td>$g(Z)$</td>
</tr>
<tr>
<td>Output</td>
<td>$g(Z)$</td>
</tr>
</tbody>
</table>

Like threshold neurons, perceptron inputs can be either the initial raw data inputs or the output from another perceptron. The primary purpose of the network training is to estimate the weights associated with each perceptron’s potential. The activation function maps this potential to the perceptron’s output.

**The Activation Function**

Although theoretically any differential function can be used as an activation function, the identity and sigmoid functions are the two most commonly used.

The *identity activation* function, also referred to as a *linear activation* function, is a flow-through mapping of the perceptron’s potential to its output:

$$g(Z) = Z$$

Output perceptrons in a forecasting network often use the identity activation function.
If the identity activation function is used throughout the network, then it is easily shown that the network is equivalent to fitting a linear regression model of the form

\[ Y_i = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k, \]

where \( x_1, x_2, \ldots, x_k \) are the \( k \) network inputs, \( Y_i \) is the \( i \)-th network output and \( \beta_0, \beta_1, \ldots, \beta_k \) are the coefficients in the regression equation. As a result, it is uncommon to find a neural network with identity activation used in all its perceptrons.

**Sigmoid activation** functions are differentiable functions that map the perceptron’s potential to a range of values, such as 0 to 1, i.e., \( \mathbb{R}^K \rightarrow \mathbb{R} \) where \( K \) is the number of perceptron inputs.
In practice, the most common sigmoid activation function is the logistic function that maps the potential into the range 0 to 1:

\[ g(Z) = \frac{1}{1 + e^{-Z}} \]

Since \( 0 < g(Z) < 1 \), the logistic function is very popular for use in networks that output probabilities. Other popular sigmoid activation functions include:

1. the hyperbolic-tangent \( g(Z) = \tanh(Z) = \frac{e^{az} - e^{-az}}{e^{az} + e^{-az}} \)
2. the arc-tangent \( g(Z) = \frac{2}{\pi} \arctan \left( \frac{az}{2} \right) \), and
3. the squash activation function (Elliott (1993)) \( g(Z) = \frac{Z}{1 + |Z|} \)

It is easy to show that the hyperbolic-tangent and logistic activation functions are linearly related. Consequently, forecasts produced using logistic activation should be close to those produced using hyperbolic-tangent activation. However, one function may be preferred over the other when training performance is a concern. Researchers report that the training time using the hyperbolic-tangent activation function is shorter than using the logistic activation function.
Network Applications

Forecasting using Neural Networks

There are many good statistical forecasting tools. Most require assumptions about the relationship between the variables being forecasted and the variables used to produce the forecast, as well as the distribution of forecast errors. Such statistical tools are referred to as parametric methods. ARIMA time series models, for example, assume that the time series is stationary, that the errors in the forecasts follow a particular ARIMA model, and that the probability distribution for the residual errors is Gaussian, see Box and Jenkins (1970). If these assumptions are invalid, then ARIMA time series forecasts can be very poor.

Neural networks, on the other hand, require few assumptions. Since neural networks can approximate highly non-linear functions, they can be applied without an extensive analysis of underlying assumptions.

Another advantage of neural networks over ARIMA modeling is the number of observations needed to produce a reliable forecast. ARIMA models generally require 50 or more equally spaced, sequential observations in time. In many cases, neural networks can also provide adequate forecasts with fewer observations by incorporating exogenous, or external, variables in the network’s input.

For example, a company applying ARIMA time series analysis to forecast business expenses would normally require each of its departments, and each sub-group within each department to prepare its own forecast. For large corporations this can require fitting hundreds or even thousands of ARIMA models. With a neural network approach, the department and sub-group information could be incorporated into the network as exogenous variables. Although this can significantly increase the network’s training time, the result would be a single model for predicting expenses within all departments and sub-departments.

Linear least squares models are also popular statistical forecasting tools. These methods range from simple linear regression for predicting a single quantitative outcome to logistic regression for estimating probabilities associated with categorical outcomes. It is easy to show that simple linear least squares forecasts and logistic regression forecasts are equivalent to a feed-forward network with a single layer. For this reason, single-layer feed-forward networks are rarely used for forecasting. Instead multilayer networks are used.

Hutchinson (1994) and Masters (1995) describe using multilayer feed-forward neural networks for forecasting. Multilayer feed-forward networks are characterized by the forward-only flow of information in the network. The flow of information and computations in a feed-forward network is always in one direction, mapping an M-dimensional vector of inputs to a C-dimensional vector of outputs, i.e., \( \mathbb{R}^M \rightarrow \mathbb{R}^C \).

There are many other types of networks without this feed-forward requirement. Information and computations in a recurrent neural network, for example, flows in both directions. Output from one level of a recurrent neural network can be fed back, with some delay, as input into the same network, see Figure 2. Recurrent networks are very useful for time series prediction, see Mandic and Chambers (2001).

Pattern Recognition using Neural Networks

Neural networks are also extensively used in statistical pattern recognition. Pattern recognition applications that make wide use of neural networks include:

IMSL C# Numerical Library
The interest in pattern recognition using neural networks has stimulated the development of important variations of feed-forward networks. Two of the most popular are:

- Self-Organizing Maps, also called Kohonen Networks, Kohonen (1995),

Good mathematical descriptions of the neural network methods underlying these applications are given by Bishop (1995), Ripley (1996), Mandic and Chambers (2001), and Abe (2001). An excellent overview of neural networks, from a statistical viewpoint, is also found in Warner and Misra (1996).

**Neural Networks for Classification**

Classifying observations using prior concomitant information is possibly the most popular application of neural networks. Data classification problems abound in business and research. When decisions based upon data are needed, they can often be treated as a neural network data classification problem. Decisions to buy, sell, hold or do nothing with a stock, are decisions involving four choices. Classifying loan applicants as good or bad credit risks, based upon their application, is a classification problem involving two choices. Neural networks are powerful tools for making decisions or choices based upon data.

These same tools are ideally suited for automatic selection or decision-making. Incoming email, for example, can be examined to separate spam from important email using a neural network trained for this task. A good overview of solving classification problems using multilayer feed-forward neural networks is found in Abe (2001) and Bishop (1995).

There are two popular methods for solving data classification problems using multilayer feed-forward neural networks, depending upon the number of choices (classes) in the classification problem. If the classification problem involves only two choices, then it can be solved using a neural network with one logistic output. This output estimates the probability that the input data belong to one of the two choices. For example, a multilayer feed-forward network with a single, logistic output can be used to determine whether a new customer is credit-worthy. The network’s input would consist of information on the applicants credit application, such as age, income, etc. If the network output probability is above some threshold value (such as 0.5 or higher) then the applicant’s credit application is approved.

This is referred to as binary classification using a multilayer feed-forward neural network. If more than two classes are involved then a different approach is needed. A popular approach is to assign logistic output perceptrons to each class in the classification problem. The network assigns each input pattern to the class associated with the output perceptron that has the highest probability for that input pattern. However, this approach produces invalid probabilities since the sum of the individual class probabilities

*Neural Nets*
for each input is not equal to one, which is a requirement for any valid multivariate probability distribution.

To avoid this problem, the softmax activation function, see Bridle (1990), applied to the network outputs ensures that the outputs conform to the mathematical requirements of multivariate classification probabilities. If the classification problem has C categories, or classes, then each category is modeled by one of the network outputs. If $Z_i$ is the weighted sum of products between its weights and inputs for the $i$-th output, i.e., $Z_i = \sum_j w_{ji} y_{ji}$, then

$$\text{softmax}_i = \frac{e^{Z_i}}{\sum_{j=1}^C e^{Z_j}}$$

The softmax activation function ensures that the outputs all conform to the requirements for multivariate probabilities. That is,

$$0 < \text{softmax}_i < 1, \text{ for all } i = 1, 2, \ldots, C$$

and

$$\sum_{i=1}^C \text{softmax}_i = 1$$

A pattern is assigned to the $i$-th classification when $\text{softmax}_i$ is the largest among all C classes.

However, multilayer feed-forward neural networks are only one of several popular methods for solving classification problems. Others include:

- Support Vector Machines (SVM Neural Networks), Abe (2001),
- Classification and Regression Trees (CART), Breiman, et al. (1984),
- Quinlan’s classification algorithms C4.5 and C5.0, Quinlan (1993), and
- Quick, Unbiased and Efficient Statistical Trees (QUEST), Loh and Shih (1997).

Support Vector Machines are simple modifications of traditional multilayer feed-forward neural networks (MLFF) configured for pattern classification.

**Multilayer Feed-Forward Neural Networks**

A multilayer feed-forward neural network is an interconnection of perceptrons in which data and calculations flow in a single direction, from the input data to the outputs. The number of layers in a neural network is the number of layers of perceptrons. The simplest neural network is one with a single input layer and an output layer of perceptrons. The network in Figure 7 illustrates this type of network. Technically this is referred to as a one-layer feed-forward network with two outputs because the output layer is the only layer with an activation calculation.
Figure 7. A Single-Layer Feed-Forward Neural Net

In this single-layer feed-forward neural network, the network inputs are directly connected to the output layer perceptrons, $Z_1$ and $Z_2$.

The output perceptrons use activation functions, $g_1$ and $g_2$, to produce the outputs $Y_1$ and $Y_2$.

Since

$$Z_1 = \sum_{i=1}^{3} W_{1,i}X_i - \mu_1$$

and

$$Z_2 = \sum_{i=1}^{3} W_{2,i}X_i - \mu_2$$

$$Y_1 = g_1(Z_1) = g_1(\sum_{i=1}^{3} W_{1,i}X_i - \mu_1)$$

and

$$Y_2 = g_2(Z_2) = g_2(\sum_{i=1}^{3} W_{2,i}X_i - \mu_2)$$

When the activation functions $g_1$ and $g_2$ are identity activation functions, a single-layer neural net is equivalent to a linear regression model. Similarly, if $g_1$ and $g_2$ are logistic activation functions, then the single-layer neural net is equivalent to logistic regression. Because of this correspondence between single-layer neural networks and linear and logistic regression, single-layer neural networks are rarely used in place of linear and logistic regression.
The next most complicated neural network is one with two layers. This extra layer is referred to as a hidden layer. In general, there is no restriction on the number of hidden layers. However, it has been shown mathematically that a two-layer neural network, such as shown in Figure 1, can accurately reproduce any differentiable function, provided the number of perceptrons in the hidden layer is unlimited.

However, increasing the number of neurons increases the number of weights that must be estimated in the network, which in turn increases the execution time for this network. Instead of increasing the number of perceptrons in the hidden layers to improve accuracy, it is sometimes better to add additional hidden layers, which typically reduces both the total number of network weights and the computational time. However, in practice, it is uncommon to see neural networks with more than two or three hidden layers.

**Neural Network Error Calculations**

**Error Calculations for Forecasting**

The error calculations used to train a neural network are very important. Researchers have investigated many error calculations, trying to find a calculation with a short training time that is appropriate for the network’s application. Typically, error calculations are very different depending primarily on the network’s application.

For forecasting, the most popular error function is the sum-of-squared errors, or one of its scaled versions. This is analogous to using the minimum least squares optimization criterion in linear regression. Like least squares, the sum-of-squared errors is calculated by looking at the squared difference between what the network predicts for each training pattern and the target value, or observed value, for that pattern. Formally, the equation is the same as one-half the traditional least squares error:

\[
E = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{C} (t_{ij} - \hat{t}_{ij})^2
\]

where N is the total number of training cases, C is equal to the number of network outputs, \( t_{ij} \) is the observed output for the \( i \)-th training case and the \( j \)-th network output, and \( \hat{t}_{ij} \) is the network’s forecast for that case.

Common practice recommends fitting a different network for each forecast variable. That is, the recommended practice is to use \( C=1 \) when using a multilayer feed-forward neural network for forecasting. For classification problems with more than two classes, it is common to associate one output with each classification category, i.e., \( C=\text{number of classes} \).

Notice that in ordinary least squares, the sum-of-squared errors are not multiplied by one-half. Although this has no impact on the final solution, it significantly reduces the number of computations required during training.

Also note that as the number of training patterns increases, the sum-of-squared errors increases. As a result, it is often useful to use the root-mean-square (RMS) error instead of the unscaled sum-of-squared
errors:

\[ E^{\text{RMS}} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{C} (t_{ij} - \hat{t}_{ij})^2}{\sum_{i=1}^{N} \sum_{j=1}^{C} (t_{ij} - \bar{t})^2} \]

where \( \bar{t} \) is the average output:

\[ \bar{t} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{C} t_{ij}}{N \cdot C} \]

Unlike the unscaled sum-of-squared errors, \( E^{\text{RMS}} \) does not increase as \( N \) increases. The smaller the value of \( E^{\text{RMS}} \) the closer the network is predicting its targets during training. A value of \( E^{\text{RMS}} = 0 \) indicates that the network is able to predict every pattern exactly. A value of \( E^{\text{RMS}} = 1 \) indicates that the network is predicting the training cases only as well as using the mean of the training cases for forecasting.

Notice that the root-mean-squared error is related to the sum-of-squared error by a simple scale factor:

\[ E^{\text{RMS}} = \frac{2}{\bar{t}} \cdot E \]

Another popular error calculation for forecasting from a neural network is the Minkowski-R error. The sum-of-squared error, \( E \), and the root-mean-squared error, \( E^{\text{RMS}} \), are both theoretically motivated by assuming the noise in the target data is Gaussian. In many cases, this assumption is invalid. A generalization of the Gaussian distribution to other distributions gives the following error function, referred to as the Minkowski-R error:

\[ E^{R} = \sum_{i=1}^{N} \sum_{j=1}^{C} |t_{ij} - \hat{t}_{ij}|^R. \]

Notice that \( E^{R} = 2E \) when \( R = 2 \).

A good motivation for using \( E^{R} \) instead of \( E \) is to reduce the impact of outliers in the training data. The usual error measures, \( E \) and \( E^{\text{RMS}} \), emphasize larger differences between the training data and network forecasts since they square those differences. If outliers are expected, then it is better to de-emphasize larger differences. This can be done by using the Minkowski-R error with \( R=1 \). When \( R=1 \), the Minkowski-R error simplifies to the sum of absolute differences:

\[ L = E^1 = \sum_{i=1}^{N} \sum_{j=1}^{C} |t_{ij} - \hat{t}_{ij}|. \]

\( L \) is also referred to as the Laplacian error. Its name is derived from the fact that it can be theoretically justified by assuming the noise in the training data follows a Laplacian rather than Gaussian distribution.

Of course, similar to \( E \), \( L \) generally increases when the number of training cases increases. Similar to \( E^{\text{RMS}} \), a scaled version of the Laplacian error can be calculated using the following formula:

\[ L^{\text{RMS}} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{C} |t_{ij} - \hat{t}_{ij}|}{\sum_{i=1}^{N} \sum_{j=1}^{C} |t_{ij} - \bar{t}|} \]
Cross-Entropy Error for Binary Classification

As previously mentioned, multilayer feed-forward neural networks can be used for both forecasting and classification applications. Training a forecasting network involves finding the network weights that minimize either the Gaussian or Laplacian distributions, $E$ or $L$ respectively, or equivalently their scaled versions, $E_{RMS}$ or $L_{RMS}$. Although these error calculations can be adapted for use in classification by setting the target classification variable to zeros and ones, this is not recommended. Use of the sum-of-squared and Laplacian error calculations is based on the assumption that the target variable is continuous. In classification applications, the target variable is a discrete random variable with $C$ possible values, where $C$=number of classes.

A multilayer feed-forward neural network for classifying patterns into one of only two categories is referred to as a binary classification network. It has a single output: the estimated probability that the input pattern belongs to one of the two categories. The probably that it belongs to the other category is equal to one minus this probability, i.e.,

$$P(C_2) = P(\text{not } C_1) = 1 - P(C_1)$$

Binary classification applications are very common. Any problem requiring yes/no classification is a binary classification application. For example, deciding to sell or buy a stock is a binary classification problem. Deciding to approve a loan application is also a binary classification problem. Deciding whether to approve a new drug or to provide one of two medical treatments are binary classification problems.

For binary classification problems, only a single output is used, $C=1$. This output represents the probability that the training case should be classified as yes. A common choice for the activation function of the output of a binary classification networks is the logistic activation function, which always results in an output in the range 0 to 1, regardless of the perceptron’s potential.

One choice for training a binary classification network is to use sum-of-squared errors with the class value of yes patterns coded as a 1 and the no classes coded as a 0, i.e.:

$$t_{ij} = \begin{cases} 
1 & \text{if training pattern } i=\text{yes} \\
0 & \text{if the training pattern } i=\text{no}
\end{cases}$$

However, using either the sum-of-squared or Laplacian errors for training a network with these target values assumes that the noise in the training data are Gaussian. In binary classification, the zeros and ones are not Gaussian. They follow the Bernoulli distribution:

$$P(t_i = t) = p^t (1-p)^{1-t}$$

where $p$ is equal to the probability that a randomly selected case belongs to the yes class.

Modeling the binary classes as Bernoulli observations leads to the cross-entropy error function described by Hopfield (1987) and Bishop (1995):

$$E^C = - \sum_{i=1}^{N} \{ t_i \ln(\hat{t}_i) + (1-t_i) \ln(1-\hat{t}_i) \}$$

where $N$ is the number of training patterns, $t_i$ is the target value for the $i$-th case (either 1 or 0), and $\hat{t}_i$ is the network’s output for the $i$-th case. This is equal to the neural network’s estimate of the probability that the $i$-th case should be classified as yes.
For situations in which the target variable is a probability in the range \(0 < t_{ij} < 1\), the value of the cross-entropy at the networks optimum is equal to:

\[
E_{\text{min}}^C = -\sum_{i=1}^{N} \left\{ t_i \ln(t_i) + (1 - t_i) \ln(1 - t_i) \right\}
\]

Subtracting this from \(E^C\) gives an error term bounded below by zero, i.e., \(E^{CE} \geq 0\) where:

\[
E^{CE} = E^C - E_{\text{min}}^C = -\sum_{i=1}^{N} \left\{ t_i \ln \left( \frac{\hat{t}_i}{t_i} \right) + (1 - t_i) \ln \left( \frac{1 - \hat{t}_i}{1 - t_i} \right) \right\}
\]

This adjusted cross-entropy is normally reported when training a binary classification network where \(0 < t_{ij} < 1\). Otherwise \(E^C\), the non-adjusted cross-entropy error, is used. Small values, values near zero, would indicate that the training resulted in a network with a low error rate and that patterns are being classified correctly most of the time.

**Back-Propagation in Multilayer Feed-Forward Neural Network**

Sometimes a multilayer feed-forward neural network is referred to incorrectly as a back-propagation network. The term back-propagation does not refer to the structure or architecture of a network. Back-propagation refers to the method used during network training. More specifically, back-propagation refers to a simple method for calculating the gradient of the network, that is the first derivative of the weights in the network.

The primary objective of network training is to estimate an appropriate set of network weights based upon a training dataset. Many ways have been researched for estimating these weights, but they all involve minimizing some error function. In forecasting, the most commonly used error function is the sum-of-squared errors:

\[
E = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{C} (t_{ij} - \hat{t}_{ij})^2
\]

Training uses one of several possible optimization methods to minimize this error term. Some of the more common are: steepest descent, quasi-Newton, conjugant gradient, and many various modifications of these optimization routines.

Back-propagation is a method for calculating the first derivative, or gradient, of the error function required by some optimization methods. It is certainly not the only method for estimating the gradient. However, it is the most efficient. In fact, some will argue that the development of this method by Werbos (1974), Parket (1985), and Rumelhart, Hinton and Williams (1986) contributed to the popularity of neural network methods by significantly reducing the network training time and making it possible to train networks consisting of a large number of inputs and perceptrons.

Simply stated, back-propagation is a method for calculating the first derivative of the error function with respect to each network weight. Bishop (1995) derives and describes these calculations for the two most common forecasting error functions, the sum of squared errors and Laplacian error functions. Abe (2001) gives the description for the classification error function, the cross-entropy error function. For all of these error functions, the basic formula for the first derivative of the network weight \(w_{ji}\) at the \(i\)-th
perceptron applied to the output from the $j$-th perceptron:

$$\frac{\partial E}{\partial w_{ji}} = \delta_j Z_i,$$

where $Z_i = g(a_i)$ is the output from the $i$-th perceptron after activation, and

$$\frac{\partial E}{\partial w_{ji}}$$

is the derivative for a single output and a single training pattern. The overall estimate of the first derivative of $w_{ji}$ is obtained by summing this calculation over all $N$ training patterns and $C$ network outputs.

The term back-propagation gets its name from the way the term $\delta_j$ in the back-propagation formula is calculated:

$$\delta_j = g'(a_j) \cdot \sum_k w_{kj} \delta_k,$$

where the summation is over all perceptrons that use the activation from the $j$-th perceptron, $g(a_j)$.

The derivative of the activation functions, $g'(a)$, varies among these functions, see the following table:

<table>
<thead>
<tr>
<th>Activation Function</th>
<th>$g(a)$</th>
<th>$g'(a)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$g(a) = a$</td>
<td>$g'(a) = 1$ (where $a$ is a constant)</td>
</tr>
<tr>
<td>Logistic</td>
<td>$g(a) = \frac{1}{1+e^{-a}}$</td>
<td>$g'(a) = g(a)(1-g(a))$</td>
</tr>
<tr>
<td>Hyperbolic-tangent</td>
<td>$g(a) = \tanh(a)$</td>
<td>$g'(a) = \text{sech}^2(a) = 1 - \tanh^2(a)$</td>
</tr>
<tr>
<td>Squash</td>
<td>$g(a) = \frac{a}{1+</td>
<td>a</td>
</tr>
</tbody>
</table>

**Creating a Feed Forward Network**

The following code fragment creates the feed forward neural network shown in the following figure:
Figure 8. A Three-Layer Feed-Forward Neural Net

Notice that this network is more complex than the typical feed-forward network in which all nodes from each layer are connected to every node in the next layer. This network has 6 input nodes, and they are not all connected to every node in the 1st hidden layer.

Note also that the 4 perceptrons in the 1st hidden layer are not connected to every node in the 2nd hidden layer, and the perceptrons in the 2nd hidden layer are not all connected to the two outputs.

```csharp
// Example code for creating links among network nodes
FeedForwardNetwork network = new FeedForwardNetwork();
network.InputLayer.CreateInputs(6);
network.CreateHiddenLayer().CreatePerceptrons(4);
```

Neural Nets • 1631
network.CreateHiddenLayer().CreatePerceptrons(3);
network.OutputLayer.CreatePerceptrons(2);
HiddenLayers[] hiddenLayer = network.HiddenLayers;
Node[] inputNode = network.InputLayer.Nodes;
Node[] layer1Node = hiddenLayer[0].Nodes;
Node[] layer2Node = hiddenLayer[1].Nodes;
Node[] outputNode = network.OutputLayer.Nodes;

// Create links between input nodes and 1st hidden layer
network.Link(inputNode[0], layer1Node[0]);
network.Link(inputNode[0], layer1Node[1]);
network.Link(inputNode[1], layer1Node[0]);
network.Link(inputNode[1], layer1Node[1]);
network.Link(inputNode[1], layer1Node[3]);
network.Link(inputNode[2], layer1Node[1]);
network.Link(inputNode[2], layer1Node[2]);
network.Link(inputNode[3], layer1Node[3]);
network.Link(inputNode[4], layer1Node[3]);
network.Link(inputNode[5], layer1Node[3]);

// Create links between 1st and 2nd hidden layers
network.Link(layer1Node[0], layer2Node[0]);
network.Link(layer1Node[0], layer2Node[1]);
network.Link(layer1Node[0], layer2Node[2]);
network.Link(layer1Node[1], layer2Node[0]);
network.Link(layer1Node[1], layer2Node[1]);
network.Link(layer1Node[1], layer2Node[2]);
network.Link(layer1Node[2], layer2Node[0]);
network.Link(layer1Node[2], layer2Node[2]);
network.Link(layer1Node[3], layer2Node[1]);
network.Link(layer1Node[3], layer2Node[2]);

// Create links between 2nd hidden layer and output layer
network.Link(layer2Node[0], outputNode[0]);
network.Link(layer2Node[1], outputNode[0]);
network.Link(layer2Node[2], outputNode[0]);
network.Link(layer2Node[2], outputNode[1]);

// Create link between input node[0] and output node[0]
network.Link(inputNode[0], outputNode[0]);

By default, the FeedForwardNetwork constructor creates a feed forward network with an empty input layer, no hidden layers and an empty output layer. Input nodes are created by accessing the empty input layer and creating 6 nodes within it. Two hidden layers are then created within the network using the FeedForwardNetwork.CreateHiddenLayer().CreatePerceptrons() method. Four perceptrons are created within the first hidden layer and three within the second. Output perceptrons are created by accessing the empty output layer and creating the Perceptrons within it: FeedForwardNetwork.OutputLayer.CreatePerceptrons().

Links among the input nodes and perceptrons can be created using one of several approaches. If all inputs are connected to every perceptron in the first hidden layer, and if all perceptrons are connected to every perceptron in the following layer, which is a standard architecture for feed forward networks, then a call to the FeedForwardNetwork.LinkAll() method can be used to create these links.

However, this example does not use that standard configuration. Some links are missing. In this case, the approach used is to construct individual links using the FeedForwardNetwork.Link() method. This
requires one call for every link.

An alternate approach is to first create all links and then to remove those that are not needed. The following code illustrates this approach:

```
// ***************************************************************
// EXAMPLE CODE FOR REMOVING LINKS AMONG NETWORK NODES
// ***************************************************************

FeedForwardNetwork network = new FeedForwardNetwork();
InputNode[] inputNode = network.InputLayer.CreateInputs(6);
Perceptron[] hiddenLayer1 =
    network.CreateHiddenLayer().CreatePerceptrons(4);
Perceptron[] hiddenLayer2 =
    network.CreateHiddenLayer().CreatePerceptrons(3);
Perceptron[] outputLayer = network.OutputLayer.CreatePerceptrons(2);
network.LinkAll(); // Creates standard feed forward configuration

// Remove links between input nodes and 1st hidden layer
network.Remove(network.FindLink(inputNode[0],hiddenLayer1[2]));
network.Remove(network.FindLink(inputNode[0],hiddenLayer1[3]));
network.Remove(network.FindLink(inputNode[1],hiddenLayer1[3]));
network.Remove(network.FindLink(inputNode[2],hiddenLayer1[0]));
network.Remove(network.FindLink(inputNode[3],hiddenLayer1[0]));
network.Remove(network.FindLink(inputNode[3],hiddenLayer1[3]));
network.Remove(network.FindLink(inputNode[4],hiddenLayer1[0]));
network.Remove(network.FindLink(inputNode[4],hiddenLayer1[1]));
network.Remove(network.FindLink(inputNode[4],hiddenLayer1[2]));
network.Remove(network.FindLink(inputNode[5],hiddenLayer1[0]));
network.Remove(network.FindLink(inputNode[5],hiddenLayer1[1]));
network.Remove(network.FindLink(inputNode[5],hiddenLayer1[2]));

// Remove links between 1st and 2nd hidden layers
network.Remove(network.FindLink(hiddenLayer1[2],hiddenLayer2[0]));
network.Remove(network.FindLink(hiddenLayer1[3],hiddenLayer2[0]));

// Remove links between 2nd hidden layer and the output layer
network.Remove(network.FindLink(hiddenLayer2[0],outputLayer[1]));

// Add link from input node[0] to output node[0]
network.Link(inputNode[0], outputNode[0]);

// ***************************************************************
```

In the above fragment, all links are created using the FeedForwardNetwork.LinkAll() method. This creates a total of 6*4+4*3+3*2=42 links, not including the link between the first input node and the first output node. Links that skip layers are not created by the LinkAll() method.

Links are then selectively removed starting with the first input node and proceeding to links between the last hidden layer and the output layers. In this case, there are 6*4=24 possible links between the input nodes and first hidden layer. Fourteenth of them had to be removed. Between the first hidden layer and second, there are 4*3=12 possible links. Two of them were removed. Between the second hidden layer and output layer there are 3*2=6 possible links, and only one needed to be removed. Finally the skip-layer link between the first input node and first output node is added.

After creating and removing links among layers, the activation function used with each perceptron can be selected. By default, every perceptron in the hidden layers use the logistic activation function and
every perceptron in the output layers uses the linear activation function. The following fragment shows
how to change the activation function in the hidden layer perceptrons from logistic to hyperbolic-tangent
and the output layer from linear to logistic. It also creates a connection directly from the first input node
to the output node.

```
// **************************************************************
// EXAMPLE CODE FOR SETTING NON-DEFAULT ACTIVATION FUNCTIONS
// **************************************************************

FeedForwardNetwork network = new FeedForwardNetwork();
InputNode[] inputNode = network.InputLayer.CreateInputs(6);
Perceptron[] hiddenLayer1 =
    network.CreateHiddenLayer().CreatePerceptrons(4);
Perceptron[] hiddenLayer2 =
    network.CreateHiddenLayer().CreatePerceptrons(3);
Perceptron[] outputLayer =
    network.OutputLayer.CreatePerceptrons(2);

// Get Network Perceptrons for Setting Their Activation Functions
Perceptron[] perceptrons = network.Perceptrons;

for (int k = 0; k < hiddenLayer1.Length - 1; k++) {
    perceptrons[k].Activation = Imsl.DataMining.Neural.Activation.Tanh;
}
perceptrons[perceptrons.Length - 1].Activation =
    Imsl.DataMining.Neural.Activation.Logistic;
```

### Training

Trainers are used to find the network weights that produce network outputs matching a set of training
targets. The training targets together with their associated network inputs are referred to as training
patterns. Training patterns can be historical data relating network inputs to its outputs, or they can be
developed from expert opinion or theoretical analysis. In the end, each training pattern relates specific
network inputs to its real or desired target outputs.

In IMSL C# Numerical Library all trainers implement the Imsl.DataMining.Neural.ITrainer
interface. The number of training input attributes must equal the number of input nodes, and the number
of training outputs, sometimes called training targets, must equal the number of output perceptrons
created for the network.

### Single Stage Trainers

QuasiNewtonTrainer and LeastSquaresTrainer are single stage trainers. They use all available
training patterns and a specific optimization method to find optimum network weights. The best set of
weights is a set that minimizes the error between the network output and its training targets. The
following code fragment illustrates how to use the quasi-Newton method for single stage network
training.

```
// **************************************************************
// EXAMPLE CODE FOR ONE-STAGE TRAINER
```
In this example, xData and yData are two-dimensional arrays containing the input attributes and output targets respectively. The number of rows in these arrays is equal to the number of training patterns. The number of columns in xData is equal to the number of input attributes, after applying any necessary preprocessing. The number of columns in yData is equal to the number of network outputs. The GradientTolerance property is one of several optional settings for tailoring the convergence criteria used with the training optimizer.

LeastSquaresTrainer is another single stage trainer. There are two principal differences between this trainer and the quasi-Newton trainer. First their optimization algorithms are different. The least squares trainer uses the Levenberg-Marquardt algorithm to optimize the network. As the name implies, the quasi-Newton trainer uses a modified Newton algorithm for optimization. In some applications, depending upon the data and the network architecture, one method may train the network faster than the other.

Another key difference between these single stage trainers is that the least squares trainer only uses one error function, the sum of squared errors. The quasi-Newton trainer, by default, uses the same error function. However, it also has an interface that accepts a user-supplied error function.

**Multistage Trainers**

When there are a large number of training patterns, single stage trainers will often take too long to complete network training. For these applications, a multistage trainer could be used to reduce training time. Multistage trainers provide considerably more flexibility in designing an optimum training scheme. All of these trainers break network training into two stages. Stage II is optional. That is, a multistage trainer can be requested to only conduct Stage I training, or it can be requested to conduct both Stage I and II training.

The main difference between Stage I and II training is that Stage I training is conducted multiple times using randomly selected subsets of all available training patterns. Each training session is referred to as an epoch. Although each epoch uses a different set of randomly selected training patterns, the number of patterns is the same for every epoch. Typically, because they are using different data, the solutions vary among epochs.

Stage II training is conducted following the Stage I training using the best set of weights obtained during Stage I. This ensures that the weights developed during Stage II training will always be as good as or better than those determined during Stage I training. The entire set of original training patterns is used during Stage II training, and only one training session is completed.

There is no requirement to use the same trainer for both stages, although there is nothing wrong with that approach. The least squares trainer might be used for Stage I training and the quasi-Newton trainer might be used for Stage II training. In addition, the optimization settings for each trainer can be...
different. The multistage trainer is implemented using the EpochTrainer class.

The following code fragment illustrates the use of the epoch multistage trainer:

```csharp
// ***************************************************************
// EXAMPLE CODE FOR MULTISTAGE EPOCH TRAINER
// ***************************************************************
double xData[,] = ...;
double yData[,] = ...;
QuasiNewtonTrainer stageITrainer = new QuasiNewtonTrainer();
LeastSquaresTrainer stageIITrainer = new LeastSquaresTrainer();
EpochTrainer trainer = new EpochTrainer(stageITrainer, stageIITrainer);
trainer.NumberOfEpochs = 20;
trainer.EpochSize = 3000;
```

In this example, a quasi-Newton trainer is selected for the Stage I trainer, and the least squares trainer is used for Stage II. Stage I will consist of 20 training epochs. The training of each epoch uses 3,000 randomly selected training patterns with the quasi-Newton trainer. The epoch with the smallest training error supplies the starting values for the Stage II trainer.

### Data Preprocessing

Data preprocessing, or filtering, is the term used to describe the process of scaling or transforming input attributes into numerical values suitable for network training. In general it is important to scale all input attributes to a common range, either [0, 1] or [-1, 1]. The algorithm used for obtaining values for the network weights assumes that the inputs are scaled to one of these ranges. If some network inputs have values that cover a much broader range, then the initial weights can be far from optimum causing network training to fail or take an excessively long time.

Network input data are classified into three general categories: continuous, ordinal and nominal. IMSL C# Numerical Library provides methods for preprocessing all three data types. Continuous data are scaled using the ScaleFilter class. In addition, lagged versions of continuous time series data can be created using the TimeSeriesFilter or TimeSeriesClassFilter class.

Categorical data, such as color or preference ratings, are either ordinal and nominal data. UnsupervisedOrdinalFilter and UnsupervisedNominalFilter are provided to preprocess ordinal and nominal data respectively. UnsupervisedOrdinalFilter transforms ordinal data into values between 0 and 1, which allows them to be treated as continuous data.

Nominal data, on the other hand, can be transformed using several methods. UnsupervisedNominalFilter converts a single nominal variable with $m$ classes into $m$ columns containing the values 0 and 1. This is referred to as binary encoding of nominal classification information.

The following code fragment illustrates the use of some of these preprocessing methods:

```csharp
// ***************************************************************
// EXAMPLE CODE FOR PREPROCESSING NOMINAL AND CONTINUOUS DATA
// ***************************************************************
```
double[,] yData = {...};
int[] nominalVariable={.....};
int nClasses = 3;

// Create a nominal filter for binary encoding of a nominal variable
// that has 3 categorical values
UnsupervisedNominalFilter nominalFilter =
    new UnsupervisedNominalFilter(nClasses);
int[,] binaryColumns = nominalFilter.Encode(nominalVariable);

// Create a scale filter for scaling continuous data in a range of [0,1]
ScaleFilter scaleFilter = new ScaleFilter(ScaleFilter.ScalingMethod.Bounded);
// Apply the scale filter to two continuous variables, x1 and x2
scaleFilter.SetBounds(-200,1000,0,1); // Original values [-200, 1000]
scaleFilter.Encode(x1);
scaleFilter.SetBounds(0,5000,0,1); // Original values [0, 5000]
scaleFilter.Encode(x2);

// Load the encoded columns into xData
int n = nominalVariable.Length;
double[,] xData = new double[n, 3+3];
for(int i=0; i < n; i++){
    xData[i,0] = x1[i];
    xData[i,1] = x2[i];
    for(int j=0; j < nClasses; j++) xData[i,j+2] = binaryColumns[i,j];
}

In the above example, one nominal variable consisting of values representing 3 different classes, or
categories, is encoded into 3 binary columns using UnsupervisedNominalFilter class. Two
continuous variables are scaled using the ScaleFilter class, and these five columns are then loaded
into xData in preparation for network training.

**Serialization**

Neural network training can require a substantial amount of time, so it is often desirable to save a trained
network for later use in forecasting. Serialization can be used to save the results of network training.

When an object is serialized, its state is saved. However, the code implementing the class (the class file)
is not saved with the serialized file. Hence when the object is deserialized, the code that created the
serialized object should be in the classpath. Otherwise deserialization will fail.

For an object to be serialized, the class must use the `Serializable` attribute. The following code fragment
serializes key network and training information into four files. One contains the network weights,
another contains the training statistics, and two additional files contain the training patterns. This is done
using a `write(Object,String)` method that takes a file name and writes the serialized object to that
file.

```java
// **************************************************************
// EXAMPLE CODE FOR SAVING TRAINED NETWORK USING SERIALIZATION
// **************************************************************
```
using System.Runtime.Serialization;

// ***************************************************************
// SAVE A TRAINED NETWORK BY SAVING THE SERIALIZED NETWORK OBJECTS
// ***************************************************************
// Saving network weights and structural information
write(network, "MyNetwork.ser");
// Saving training information available from computeStatistics()
write(trainer, "MyNetworkTrainer.ser");
// Saving xData training targets
write(xData, "MyNetworkxData.ser");
// Saving yData training targets
write(yData, "MyNetworkyData.ser");

// **************************************************************************
// WRITE SERIALIZED NETWORK TO A FILE
// **************************************************************************
static public void write(System.Object obj, System.String filename)
{
    System.IO.FileStream fos = new System.IO.FileStream(filename,
        System.IO.FileMode.Create);
    IFormatter oos = new BinaryFormatter();
    oos.Serialize(fos, obj);
    fos.Close();
}

// ***************************************************************
Notice that not only is the network object serialized and saved, the trainer and training patterns, xData
and yData, are also saved. This was only done to allow someone to calculate the additional network
statistics. If these are not needed, then these training patterns need not be saved. However, for
forecasting, it is essential to remember the specific order and nature of the network inputs used during
training. This information is not saved in the network serialized file.

When an object is deserialized, the object is reconstructed using the saved serialization file. The
following code deserializes the previously saved network information.

// **************************************************************************
// EXAMPLE CODE FOR READING TRAINED NETWORK FROM SERIALIZED FILES
// **************************************************************************
using System.Runtime.Serialization;

// READ THE TRAINED NETWORK FROM THE SERIALIZE NETWORK OBJECT
Network network = (Network) read("MyNetwork.ser");
// READ THE SERIALIZE XDATA[,] AND YDATA[,] ARRAYS OF TRAINING
// PATTERNS.
xData = (double[,,]) read("MyNetworkxData.ser");
yData = (double[,])read("MyNetworkyData.ser");
// READ THE SERIALIZED TRAINER OBJECT
Trainer trainer = (ITrainer)read("MyNetworkTrainer.ser");
// **************************************************************
// DISPLAY TRAINING STATISTICS
// **************************************************************
double stats[] = network.computeStatistics(xData, yData);

// **************************************************************
// READ SERIALIZED NETWORK FROM A FILE
// **************************************************************
static public System.Object read(System.String filename)
{
    System.IO.FileStream fis = new System.IO.FileStream(filename, System.IO.FileMode.Open, System.IO.FileAccess.Read);
    IFormatter ois = new BinaryFormatter();
    System.Object obj = (System.Object) ois.Deserialize(fis);
    fis.Close();
    return obj;
}
// **************************************************************

FeedForwardNetwork Class

public class Imsl.DataMining.Neural.FeedForwardNetwork : Network

A representation of a feed forward neural network.

A Network contains an InputLayer (p. 1640), an OutputLayer (p. 1641) and zero or more HiddenLayers (p. 1656). The null InputLayer and OutputLayer are automatically created by the Network (p. 1786) constructor. The InputNodes are added using the InputLayer.CreateInputs(nInputs) method. Output Perceptrons (p. 1661) are added using the OutputLayer.CreatePerceptrons(nOutputs) method, and HiddenLayers can be created using the CreateHiddenLayer().CreatePerceptrons(nPerceptrons) method.

The InputLayer contains InputNodes. The HiddenLayers and OutputLayers contain Perceptron nodes. These Nodes (p. 1659) are created using factory methods in the Layers (p. 1654).

The Network also contains Links (p. 1643) between Nodes. Links are created by methods in this class.

Each Link has a weight and gradient value. Each Perceptron node has a bias value. When the Network is trained, the weight and bias values are used as initial guesses. After the Network is trained the weight, gradient and bias values are set to the values computed by the training.

A feed forward network is a network in which links are only allowed from one layer to a following layer.
Properties

HiddenLayers
virtual public Imsl.DataMining.Neural.HiddenLayer[] HiddenLayers {get; }

Description
The HiddenLayers in this Network.

Property Value
An array of HiddenLayers in this Network.

InputLayer
override public Imsl.DataMining.Neural.InputLayer InputLayer {get; }

Description
The InputLayer in this Network.

Property Value
The neural network InputLayer.

Links
override public Imsl.DataMining.Neural.Link[] Links {get; }

Description
All the Links in this Network.

Property Value
An array of Links containing all of the Links in this Network.

NumberOfInputs
override public int NumberOfInputs {get; }

Description
The number of InputNodes to the Network.

Property Value
An int containing the number of InputNodes to the Network.

NumberOfLinks
override public int NumberOfLinks {get; }

Description
The number of Links in the Network.

Property Value
An int which contains the number of Links in the Network.

NumberOfOutputs
override public int NumberOfOutputs {get; }

1640 • FeedForwardNetwork

IMSL C# Numerical Library
**Description**
The number of output Perceptrons from the Network.

**Property Value**
An int containing the number of output Perceptrons from the Network.

**NumberOfWeights**
override public int NumberOfWeights {get; }

**Description**
The number of *weights* in the Network.

**Property Value**
An int which contains the number of *weights* in the Network.

**OutputLayer**
override public Imsl.DataMining.Neural.OutputLayer OutputLayer {get; }

**Description**
The neural network *OutputLayer*.

**Property Value**
Contains the neural network *OutputLayer*.

**Perceptrons**
override public Imsl.DataMining.Neural.Perceptron[] Perceptrons {get; }

**Description**
The *Perceptrons* in this Network.

**Property Value**
An array of *Perceptrons* in this Network.

**Weights**
override public double[] Weights {get; set; }

**Description**
The *weight* values for the Links (p. 1666) in this Network.

**Property Value**
An array of doubles containing the *weights*.

**Remarks**
The array contains the *weights* for each Link followed by the Perceptron (p. 1662) *bias* values. The Link *weights* are the order in which the Links were created. The *weights* values are first, followed by the *bias* values in the HiddenLayer and then the *bias* values in the OutputLayer, and in the order in which the Perceptrons were created.
Constructor

**FeedForwardNetwork**

```csharp
public FeedForwardNetwork()
```

**Description**

Creates a new instance of `FeedForwardNetwork`.

Methods

**CreateHiddenLayer**

```csharp
override public Imsl.DataMining.Neural.HiddenLayer CreateHiddenLayer()
```

**Description**

Creates a `HiddenLayer`.

**Returns**

A `HiddenLayer` object which specifies a neural network hidden layer.

**FindLink**

```csharp
```

**Description**

Returns the Link between two Nodes.

**Parameters**

- `from` – The origination Node.
- `to` – The destination Node.

**Returns**

A Link between the two Nodes, or null if no such Link exists.

**FindLinks**

```csharp
virtual public Imsl.DataMining.Neural.Link[] FindLinks(Imsl.DataMining.Neural.Node to)
```

**Description**

Returns all of the Links to a given Node.

**Parameter**

- `to` – A Node whose Links are to be determined.
Returns
An array of Links containing all of the Links to the given Node.

Forecast
override public double[] Forecast(double[] x)

Description
Computes a forecast using the Network.

Parameter
x – A double array of values to which the Nodes in the InputLayer are to be set.

Returns
A double array containing the values of the Nodes in the OutputLayer.

GetForecastGradient
override public double[,] GetForecastGradient(double[] xData)

Description
Returns the derivatives of the outputs with respect to the weights.

Parameter
xData – A double array which specifies the input values at which the gradient is to be evaluated.

Returns
A double array containing the gradient values.

Remarks
The value of gradient[i][j] is dy_i/dw_j, where y_i is the i-th output and w_j is the j-th weight.

Link

Description
Establishes a Link between two Nodes.

Parameters
from – The origination Node.

to – The destination Node.

Returns
A Link between the two Nodes.
Remarks
Any existing Link between these Nodes is removed.

Link

Description
Establishes a Link between two Nodes with a specified weight (p. 1666).

Parameters
from – The origination Node.
to – The destination Node.
weight – A double which specifies the weight to be given the Link.

Returns
A Link between the two Nodes.

LinkAll
virtual public void LinkAll(Imsl.DataMining.Neural.Layer from, Imsl.DataMining.Neural.Layer to)

Description
Links all of the Nodes in one Layer to all of the Nodes in another Layer.

Parameters
from – The origination Layer.
to – The destination Layer.

LinkAll
virtual public void LinkAll()

Description
For each Layer in the Network, link each Node in the Layer to each Node in the next Layer.

Remove
virtual public void Remove(Imsl.DataMining.Neural.Link link)

Description
Removes a Link from the Network.

Parameter
link – The Link deleted from the Network.

SetEqualWeights
virtual public void SetEqualWeights(double[,] xData)
Description
Initializes network weights using equal weighting.

Parameter
xData – An input double matrix containing training patterns. The number of columns in xData must equal the number of nodes in the InputLayer.

Remarks
The equal weights approach starts by assigning equal values to the inputs of each Perceptron. If a Perceptron has 4 inputs, then this method starts by assigning the value 1/4 to each of the Perceptron’s input weights. The bias weight is initially assigned a value of zero.

The weights for the first layer of Perceptrons, either the first HiddenLayer if the number of layers is greater than 1 or the OutputLayer, are scaled using the training patterns. Scaling is accomplished by dividing the initial weights for the first layer by the standard deviation, s, of the potential for that Perceptron. The bias weight is set to -avg/s, where avg is the average potential for that Perceptron. This makes the average potential for the Perceptrons in this first layer approximately 0 and its standard deviation equal to 1.

This reduces the possibility of saturation during network training resulting from very large or small values for the Perceptrons potential. During training random noise is added to these initial values at each training stage. If the EpochTrainer is used, noise is added to these initial values at the start of each epoch.

SetRandomWeights
virtual public void SetRandomWeights(double[,] xData, System.Random random)

Description
Initializes network weights using random weights.

Parameters
xData – An input double matrix containing training patterns. The number of columns in xData must equal the number of nodes in the InputLayer.
random – A Random object.

Remarks
The random weights algorithm assigns equal weights to all Perceptrons, except those in the first layer connected to the InputLayer. Like the equal weights algorithm, Perceptrons not in the first layer are assigned weights 1/k, where k is the number of InputNodes connected to that Perceptron.

For the first layer Perceptron weights are initially assigned values from the uniform random distribution on the interval [-0.5, +0.5]. These are then scaled using the training patterns. The random weights for a perceptron are divided by s, the standard deviation of the potential for that Perceptron calculated using the initial random values. Its bias value is set to -avg/s, where avg is the average potential for that Perceptron. This makes the average potential for the Perceptrons in this first layer approximately 0 and its standard deviation equal to 1.

This reduces the possibility of saturation during network training resulting from very large or small values for the Perceptrons potential. During training random noise is added to these initial values at
each training stage. If the EpochTrainer is used, noise is added to these initial values at the start of
each epoch.

**ValidateLink**

```csharp
virtual protected internal void ValidateLink(Imsl.DataMining.Neural.Node from,
Imsl.DataMining.Neural.Node to)
```

**Description**

Checks that a Link between two Nodes is valid.

**Parameters**

- `from` – The origination Node.
- `to` – The destination Node.

**Remarks**

In a feed forward network a link must be from a node in one layer to a node in a later layer. Intermediate
layers can be skipped, but a link cannot go backward.

**Exception**

- System.ArgumentException is thrown if the Link is not valid

**Example: FeedForwardNetwork**

This example trains a 2-layer network using 100 training patterns from one nominal and one continuous
input attribute. The nominal attribute has three classifications which are encoded using binary encoding.
This results in three binary network input columns. The continuous input attribute is scaled to fall in the
interval [0,1].

The network training targets were generated using the relationship:

\[ y = 10 \times X_1 + 20 \times X_2 + 30 \times X_3 + 2.0 \times X_4, \]

where

- \( X_1 \) to \( X_3 \) are the three binary columns, corresponding to categories 1-3 of the nominal attribute, and \( X_4 \) is
the scaled continuous attribute.

The structure of the network consists of four input nodes and two layers, with three perceptrons in the
hidden layer and one in the output layer. The following figure illustrates this structure:
There are a total of 19 weights in this network. The activations functions are all linear. Since the target output is a linear function of the input attributes, linear activation functions guarantee that the network forecasts will exactly match their targets. Of course, this same result could have been obtained using linear multiple regression. Training is conducted using the quasi-newton trainer.

```csharp
using System;
using Imsl.DataMining.Neural;
using System.Runtime.Serialization;

//*****************************************************************************
// Two Layer Feed-Forward Network with 4 inputs: 1 nominal with 3 categories,
// encoded using binary encoding, 1 continuous input attribute, and 1 output
// target (continuous).
// There is a perfect linear relationship between the input and output
// variables:
// MODEL: Y = 10*X1+20*X2+30*X3+2*X4
// Variables X1-X3 are the binary encoded nominal variable and X4 is the
// continuous variable.
//*****************************************************************************

[Serializable]
```
public class FeedForwardNetworkEx1 //: System.Runtime.Serialization.ISerializable
{
    // Network Settings
    private static int nObs = 100; // number of training patterns
    private static int nInputs = 4; // four inputs
    private static int nCategorical = 3; // three categorical attributes
    private static int nOutputs = 1; // one continuous output
    private static int nPerceptrons = 3; // perceptrons in hidden layer
    private static IActivation hiddenLayerActivation;
    private static IActivation outputLayerActivation;
    private static System.String errorMsg = "";
    // Error Status Messages for the Least Squares Trainer
    private static System.String errorMsg0 = "--> Least Squares Training Completed Successfully";
    private static System.String errorMsg1 = "--> Scaled step tolerance was satisfied. The current solution \n" + "may be an approximate local solution, or the algorithm is making\n" + "slow progress and is not near a solution, or the Step Tolerance\n" + "is too big;"
    private static System.String errorMsg2 = "--> Scaled actual and predicted reductions in the function are\n" + "less than or equal to the relative function convergence\n" + "tolerance RelativeTolerance;"
    private static System.String errorMsg3 = "--> Iterates appear to be converging to a noncritical point.\n" + "Incorrect gradient information, a discontinuous function,\n" + "or stopping tolerances being too tight may be the cause.";
    private static System.String errorMsg4 = "--> Five consecutive steps with the maximum stepsize have\n" + "been taken. Either the function is unbounded below, or has\n" + "a finite asymptote in some direction, or the maximum stepsize\n" + "is too small;"
    private static System.String errorMsg5 = "--> Too many iterations required;"

    // categoricalAtt[]: A 2D matrix of values for the categorical training
    // attribute. In this example, the single categorical
    // attribute has 3 categories that are encoded using
    // binary encoding for input into the network.
    // {1,0,0} = category 1, {0,1,0} = category 2, and
    // {0,0,1} = category 3.
    private static double[,] categoricalAtt = {
        {1, 0, 0}, {1, 0, 0}, {1, 0, 0}, {1, 0, 0}, {1, 0, 0},
        {1, 0, 0}, {1, 0, 0}, {1, 0, 0}, {1, 0, 0}, {0, 1, 0},
        {0, 1, 0}, {0, 1, 0}, {0, 1, 0}, {0, 1, 0}, {0, 1, 0},
        {0, 1, 0}, {0, 1, 0}, {0, 1, 0}, {0, 1, 0}, {0, 1, 0},
        {0, 1, 0}, {0, 1, 0}, {0, 1, 0}, {0, 1, 0}, {0, 1, 0},
        {0, 1, 0}, {0, 1, 0}, {0, 1, 0}, {0, 1, 0}, {0, 1, 0},
    };
}

1648 • FeedForwardNetwork  IMSL C# Numerical Library
contAtt[]: A matrix of values for the continuous training attribute


outs[]: A 2D matrix containing the training outputs for this network
In this case there is an exact linear relationship between these
outputs and the inputs: outs = 10*X1+20*X2+30*X3+2*X4, where
X1-X3 are the categorical variables and X4=contAtt


Neural Nets FeedForwardNetwork • 1649
public static void Main(System.String[] args) {

double[] weight; // network weights
double[] gradient; // network gradient after training
double[,] xData; // Input Attributes for Trainer
double[,] yData; // Output Attributes for Trainer
int i, j; // array indices
int nWeights = 0; // Number of weights obtained from network
System.String networkFileName = "FeedForwardNetworkEx1.ser";
System.String trainerFileName = "FeedForwardTrainerEx1.ser";
System.String xDataFileName = "FeedForwardxDataEx1.ser";
System.String yDataFileName = "FeedForwardyDataEx1.ser";

// PREPROCESS TRAINING PATTERNS
System.Console.Out.WriteLine("--> Starting Preprocessing of Training Patterns");
xData = new double[nObs,nInputs];
for (int i2 = 0; i2 < nObs; i2++)
{
    xData[i2] = new double[nInputs];
}
yData = new double[nObs,nOutputs];
for (int i3 = 0; i3 < nObs; i3++)
{
    yData[i3] = new double[nOutputs];
}
for (i = 0; i < nObs; i++)
{
    for (j = 0; j < nCategorical; j++)
    {
        xData[i,j] = categoricalAtt[i,j];
    }
    xData[i,nCategorical] = contAtt[i] / 10.0; // Scale continuous input
    yData[i,0] = outs[i]; // outputs are unscaled
}

// CREATE FEEDFORWARD NETWORK
System.Console.Out.WriteLine("--> Creating Feed Forward Network Object");
FeedForwardNetwork network = new FeedForwardNetwork();
// setup input layer with number of inputs = nInputs = 4
network.InputLayer.CreateInputs(nInputs);
// create a hidden layer with nPerceptrons=3 perceptrons
network.CreateHiddenLayer().CreatePerceptrons(nPerceptrons);
// create output layer with nOutputs=1 output perceptron
network.OutputLayer.CreatePerceptrons(nOutputs);
// link all inputs and perceptrons to all perceptrons in the next layer
network.LinkAll();
// Get Network Perceptrons for Setting Their Activation Functions
Perceptron[] perceptrons = network.Perceptrons;
// Set all perceptrons to linear activation
for (i = 0; i < perceptrons.Length - 1; i++)
{
    perceptrons[i].Activation = hiddenLayerActivation;
}
perceptrons[perceptrons.Length - 1].Activation = outputLayerActivation;
System.Console.Out.WriteLine("--> Feed Forward Network Created with 2 Layers");
// **********************************************************************
// TRAIN NETWORK USING QUASI-NEWTON TRAINER
// **********************************************************************
// Create Trainer
QuasiNewtonTrainer trainer = new QuasiNewtonTrainer();
// Set Training Parameters
trainer.MaximumTrainingIterations = 1000;
// Train Network
trainer.Train(network, xData, yData);
// Check Training Error Status
switch (trainer.ErrorStatus)
{
    case 0: errorMsg = errorMsg0; break;
    case 1: errorMsg = errorMsg1; break;
    case 2: errorMsg = errorMsg2; break;
    case 3: errorMsg = errorMsg3; break;
    case 4: errorMsg = errorMsg4; break;
    case 5: errorMsg = errorMsg5; break;
    default: errorMsg = errorMsg0; break;
}
System.Console.Out.WriteLine(errorMsg);
// **********************************************************************
// DISPLAY TRAINING STATISTICS
// **********************************************************************
double[] stats = network.ComputeStatistics(xData, yData);
// Display Network Errors
System.Console.Out.WriteLine("***********************************************");
System.Console.Out.WriteLine("--> SSE: " + (float) stats[0]);
System.Console.Out.WriteLine("--> RMS: " +
System.Console.Out.WriteLine("--> Laplacian Error: " + (float) stats[1]);
System.Console.Out.WriteLine("--> Largest Absolute Residual: " + (float) stats[3]);
// Get weights
weight = network.Weights;
// Get number of weights = number of gradients
nWeights = network.NumberOfWeights;
// Obtain Gradient Vector
gradient = trainer.ErrorGradient;
// Print Network Weights and Gradients
System.Console.Out.WriteLine(" ");
System.Console.Out.WriteLine("***********************************************");
for (i = 0; i < nWeights; i++)
{
    System.Console.Out.WriteLine("w[" + i + "]=" + (float) weight[i] + " g[" + i + "]=" + (float) gradient[i]);
}
System.Console.Out.WriteLine("***********************************************");
System.Console.Out.WriteLine("n--> Saving Trained Network into " + networkFileName);
write(network, networkFileName);
System.Console.Out.WriteLine("--> Saving xData into " + xDataFileName);
write(xData, xDataFileName);
System.Console.Out.WriteLine("--> Saving yData into " + yDataFileName);
write(yData, yDataFileName);
System.Console.Out.WriteLine("--> Saving Network Trainer into " + trainerFileName);
write(trainer, trainerFileName);
}

// **********************************************************************
// WRITE SERIALIZED NETWORK TO A FILE
// **********************************************************************
static public void write(System.Object obj, System.String filename)
{
    System.IO.FileStream fos = new System.IO.FileStream(filename, System.IO.FileMode.Create);
    IFormatter oos = new BinaryFormatter();
    oos.Serialize(fos, obj);
    fos.Close();
}
static FeedForwardNetworkEx1()
{
    hiddenLayerActivation = Imsl.DataMining.Neural.Activation.Linear;
    outputLayerActivation = Imsl.DataMining.Neural.Activation.Linear;
}

Output

--> Starting Preprocessing of Training Patterns
--> Creating Feed Forward Network Object
--> Feed Forward Network Created with 2 Layers
--> Training Network using Quasi-Newton Trainer
--> Least Squares Training Completed Successfully
***********************************************************************************************
--> SSE: 1.013444E-15
--> RMS: 2.007463E-19
--> Laplacian Error: 3.005804E-07
--> Scaled Laplacian Error: 3.535235E-10
--> Largest Absolute Residual: 2.784275E-08
***********************************************************************************************

--> Getting Network Weights and Gradients

--> Network Weights and Gradients:
***********************************************************************************************
w[0]=-1.491785 g[0]=-2.611079E-08
w[1]=-1.491785 g[1]=-2.611079E-08
w[8]=4.725622 g[8]=-5.273856E-08
w[12]=1.072258 g[12]=-1.690978E-07
w[14]=1.072258 g[14]=-1.690978E-07
w[16]=3.850755 g[16]=-1.7029E-08
w[17]=3.850755 g[17]=-1.7029E-08
w[18]=2.411725 g[18]=-1.588144E-08
***********************************************************************************************

--> Saving Trained Network into FeedForwardNetworkEx1.ser
--> Saving xData into FeedForwardxDataEx1.ser
--> Saving yData into FeedForwardyDataEx1.ser
--> Saving Network Trainer into FeedForwardTrainerEx1.ser
Layer Class

public class Imsl.DataMining.Neural.Layer

The base class for Layers in a neural network.

See Also


Properties

Index

virtual public int Index {get; set; }

Description
The Index of this Layer.

Nodes

virtual public Imsl.DataMining.Neural.Node[] Nodes {get; }

Description
A list of the Nodes in this Layer.

Property Value
An array containing the Nodes associated with this Layer.

Constructor

Layer

protected internal Layer(Imsl.DataMining.Neural.FeedForwardNetwork network)

Description
Constructs a Layer.

Parameter
network – The FeedForwardNetwork to which this Layer is associated.
**Method**

**AddNode**

virtual protected internal void AddNode(Imsl.DataMining.Neural.Node node)

**Description**

Associates a Imsl.DataMining.Neural.Perceptron (p. 1661) with this Layer.

**Parameter**

node – A Node to associate with this Layer.

---

**InputLayer Class**

public class Imsl.DataMining.Neural.InputLayer : Layer

Input layer in a neural network.

An InputLayer is automatically created by Network.

**See Also**

Imsl.DataMining.Neural.Network (p. 1786)

**Property**

**Nodes**

override public Imsl.DataMining.Neural.Node[] Nodes {get; }

**Description**

The Perceptrons in the InputLayer.

**Property Value**

An InputNode array containing the Nodes in the InputLayer.

---

**Methods**

**CreateInput**

virtual public Imsl.DataMining.Neural.InputNode CreateInput()
**Description**
Creates an InputNode in the InputLayer of the neural network.

**CreateInputs**
```csharp
virtual public Imsl.DataMining.Neural.InputNode[] CreateInputs(int n)
```

**Description**
Creates a number of InputNodes in this Layer of the neural network.

**Parameter**

n – An int which specifies the number of InputNodes to be created in this Layer.

**Returns**
An InputNode array containing the created InputNodes.

---

**HiddenLayer Class**

```csharp
public class Imsl.DataMining.Neural.HiddenLayer : Layer
```

Hidden layer in a neural network. This is created by a factory method in Network.

**See Also**

Imsl.DataMining.Neural.Network.CreateHiddenLayer (p. 1789)

---

**Methods**

**CreatePerceptron**
```csharp
virtual public Imsl.DataMining.Neural.Perceptron CreatePerceptron(Imsl.DataMining.Neural.IActivation activation, double bias)
```

**Description**
Creates a Perceptron in this Layer with a specified activation function and bias.

**Parameters**

- activation – The IActivation object which specifies the activation function to be used.
- bias – A double which specifies the initial value for the bias weight.

**CreatePerceptron**
```csharp
virtual public Imsl.DataMining.Neural.Perceptron CreatePerceptron()
```
Description
Creates a Perceptron in this Layer of the neural network.

Remarks
The created Perceptron uses the logistic activation function and has an initial bias value of zero.

CreatePerceptrons
virtual public Imsl.DataMining.Neural.Perceptron[] CreatePerceptrons(int n)

Description
Creates a number of Perceptrons in this Layer of the neural network.

Parameter
n – An int which specifies the number of Perceptrons to be created.

Returns
An array containing the created Perceptrons.

Remarks
The created Perceptrons use the logistic activation function and have an initial bias value of zero.

CreatePerceptrons
virtual public Imsl.DataMining.Neural.Perceptron[] CreatePerceptrons(int n, Imsl.DataMining.Neural.IActivation activation, double bias)

Description
Creates a number of Perceptrons in this Layer with the specified bias.

Parameters
n – An int which specifies the number of Perceptrons to be created.
activation – The IActivation object which specifies the action function to be used.
bias – A double containing the initial value to be applied as the Bias values for the Perceptrons.

Returns
An array containing the created Perceptrons.

OutputLayer Class

public class Imsl.DataMining.Neural.OutputLayer : Layer
Output layer in a neural network.
An empty OutputLayer is automatically created by FeedForwardNetwork.
See Also

Imsl.DataMining.Neural.Network (p. 1786)

Property

Nodes
override public Imsl.DataMining.Neural.Node[] Nodes {get; }

Description
The Perceptrons in the OutputLayer.

Property Value
An OutputPerceptron array containing the Nodes in the OutputLayer.

Remarks
This method overrides the method in Layer (p. 1654) to return the Perceptrons in an OutputPerceptron array.

Methods

CreatePerceptron
virtual public Imsl.DataMining.Neural.Perceptron
CreatePerceptron(Imsl.DataMining.Neural.IActivation activation, double bias)

Description
Creates a Perceptron in this Layer with a specified activation and bias.

Parameters
activation – The IActivation object which specifies the action function to be used.
bias – A double which specifies the initial value for the bias for this Perceptron.

CreatePerceptron
virtual public Imsl.DataMining.Neural.Perceptron
CreatePerceptron()

Description
Creates a Perceptron in this Layer of the neural network. By default, the created Perceptron uses the linear activation function and has an initial bias value of zero.

CreatePerceptrons
virtual public Imsl.DataMining.Neural.Perceptron[]
CreatePerceptrons(int n)
Description
Creates a number of Perceptrons in this Layer of the neural network. By default, they will use linear activation and a zero initial bias.

Parameter
n – An int which specifies the number of Perceptrons to be created in this Layer.

Returns
An array containing the created Perceptrons.

CreatePerceptrons
virtual public Imsl.DataMining.Neural.Perceptron[] CreatePerceptrons(int n, Imsl.DataMining.Neural.IActivation activation, double bias)

Description
Creates a number of Perceptrons in this Layer with specified activation and bias Bias (p. 1662).

Parameters
n – An int which specifies the number of Perceptrons to be created.
activation – The IActivation object which indicates the action function to be used.
bias – A double which specifies the initial bias for the Perceptrons.

Returns
An array containing the created Perceptrons.

Node Class

public class Imsl.DataMining.Neural.Node
A Node in a neural network.
Node is an abstract class that serves as the base class for the concrete classes InputNode and Perceptron.

See Also
Property

Layer
virtual public Imsl.DataMining.Neural.Layer Layer {get; }

Description
The Layer in which this Node exists.

Property Value
The Layer associated with this Node.

Methods

GetValue
virtual public double GetValue()

Description
Returns the value of this Node.

Returns
A double which contains the value of the Node.

SetValue
virtual public void SetValue(double node)

Description
Sets the value of this Node.

Parameter
node – A double which specifies a value for the Node.

InputNode Class

public class Imsl.DataMining.Neural.InputNode : Node

A Node in the InputLayer.

InputNodes are not created directly. Instead factory methods in InputLayer are used to create InputNodes within the InputLayer. For example, InputLayer.CreateInput (p. 1655) creates a single InputNode.
Methods

GetValue
override public double GetValue()

Description
Returns the value of this Imsl.DataMining.Neural.Node (p. 1659).

Returns
A double which contains the value of this InputNode.

SetValue
override public void SetValue(double node)

Description
Sets the value of this Imsl.DataMining.Neural.Node (p. 1659).

Parameter
node – A double which specifies the new value of this InputNode.

Perceptron Class

public class Imsl.DataMining.Neural.Perceptron : Node

A Perceptron node in a neural network.

Perceptrons are created by factory methods in a Layer. Each Perceptron has an activation function \( g \) (p. 1661) and a bias \( \mu \) (p. 1662). The value of a Perceptron is given by \( g(\sum w_i X_i + \mu) \), where \( X_i \)s are the values of nodes input to this Perceptron with weight \( w_i \) (p. 1666).

Network training will use existing bias values for the starting values for the trainer. Upon completion of Network training, the bias values are set to the values computed by the trainer.

Properties

Activation
virtual public Imsl.DataMining.Neural.IActivation Activation {get; set; }

Neural Nets  Perceptron • 1661
Description
The activation function.

Property Value
An Activation object which represents the activation $g$ used by this Perceptron.

Bias
virtual public double Bias {get; set; }

Description
The *bias* for this perceptron.

Property Value
A double representing the *bias* for this Perceptron.

Remarks
The Bias has a default value of 0.

OutputPerceptron Class

```csharp
public class Imsl.DataMining.Neural.OutputPerceptron : Perceptron
{
    A Perceptron in the OutputLayer.

    OutputPerceptrons are created by factory methods in OutputLayer.
    OutputPerceptrons are not created directly. Instead factory methods in OutputLayer are used to
    create OutputPerceptrons within the OutputLayer. For example, OutputLayer.createPerceptron() creates a single OutputPerceptron.
}
```

See Also
Imsl.DataMining.Neural.OutputLayer (p. 1657)

Method

GetValue
override public double GetValue()

Description
Returns the value of the output perceptron determined using the current Network state and inputs.
Returns
A double value of the OutputPerceptron determined using the current Network state and inputs.

IActivation Interface

public interface Imsl.DataMining.Neural.IActivation

Interface implemented by perceptron activation functions.

Standard activation functions are defined as static members of this interface. New activation functions
can be defined by implementing a method, \( g(\text{double } x) \), returning the value and a method,
\( \text{derivative}(\text{double } x, \text{ double } y) \), returning the derivative of \( g \) evaluated at \( x \) where \( y = g(x) \).

See Also

Imsl.DataMining.Neural.Perceptron (p. 1661)

Methods

Derivative

abstract public double Derivative(double x, double y)

Description

Returns the value of the derivative of the activation function.

Parameters

\( x \) – A double which specifies the point at which the activation function is to be evaluated.

\( y \) – A double which specifies \( y = g(x) \), the value of the activation function at \( x \).

Returns

A double containing the value of the derivative of the activation function at \( x \).

Remarks

\( y \) is not mathematically required, but can sometimes be used to more quickly compute the derivative.

G

abstract public double G(double x)

Description

Returns the value of the activation function.
Parameter

x – A double is the point at which the activation function is to be evaluated.

Returns
A double containing the value of the activation function at x.

Activation Structure

public structure Imsl.DataMining.Neural.Activation

Fields

Linear
public Imsl.DataMining.Neural.IActivation Linear

Description
The identity activation function, \( g(x) = x \).

Logistic
public Imsl.DataMining.Neural.IActivation Logistic

Description
The logistic activation function, \( g(x) = \frac{1}{1+e^{-x}} \).

LogisticTable
public Imsl.DataMining.Neural.IActivation LogisticTable

Description
The logistic activation function computed using a table.

Remarks
This is an approximation to the logistic function that is faster to compute.
This version of the logistic function differs from the exact version by at most 4.0e-9.
Networks trained using this activation should not use Activation.Logistic for forecasting.
Forecasting should be done using the specific function supplied during training.

Softmax
public Imsl.DataMining.Neural.IActivation Softmax
**Description**

The softmax activation function.

\[
\text{softmax}_i = \frac{e^{Z_i}}{\sum_{j=1}^{C} e^{Z_j}}
\]

---

**Squash**

`public Imsl.DataMining.Neural.IActivation Squash`

**Description**

The squash activation function, \( g(x) = \frac{x}{1 + |x|} \)

---

**Tanh**

`public Imsl.DataMining.Neural.IActivation Tanh`

**Description**

The hyperbolic tangent activation function, \( g(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \).

---

**Link Class**

`public class Imsl.DataMining.Neural.Link`

A link in a neural network.

Link objects are not created directly. Instead, they are created by factory methods in `FeedForwardNetwork`.

The most useful method is `LinkAll()` (p. 1644) which creates Link objects connecting every Node in each Layer to every Node in the next Layer.

The method `Link(node,node)` (p. 1643) creates a Link from a Node to any Node in a later Layer.

The method `FindLink(Node,Node)` (p. 1642) returns the Link connecting two Nodes in the `Imsl.DataMining.Neural.Network` (p. 1786).

The method `Remove(Link)` (p. 1644) removes a Link from the Network.

Each Link object contains a `weight`, see property `Weight` (p. 1666). `Weights` are used in computing Perceptron values.

---

**See Also**

Imsl.DataMining.Neural.FeedForwardNetwork (p. 1639)
Properties

From
classvirtualpublicImsl.DataMining.Neural.NodeFrom{get;}

Description
The origination Node for this Link.

Property Value
A Node which is the origination Node for this Link.

To
classvirtualpublicImsl.DataMining.Neural.NodeTo{get;}

Description
The destination Node for this Link.

Property Value
A Node which is the destination Node for this Link.

Weight
classvirtualpublicdoubleWeight{get;set;}

Description
The weight for this Link.

Property Value
A double which contains the weight attributed to this Link.

ITrainer Interface

classpublicinterfaceImsl.DataMining.Neural.ITrainer

Interface implemented by classes used to train a Network.

The method Train is used to adjust the Weights (p. 1666) in a network to best fit a set of observed data. After a Network is trained, the other methods in this interface can be used to check the quality of the fit.

Properties

ErrorGradient
classabstractpublicdouble[]ErrorGradient{get;}

1666•ITrainerIMSL C# Numerical Library
**Description**
The value of the gradient of the error function with respect to the Weights (p. 1666).

**Property Value**
A double array, the length of the number of weights, containing the value of the gradient of the error function with respect to the weights at the computed optimal point.

**Remarks**
Before training, null is returned.

**ErrorStatus**
abstract public int ErrorStatus {get; }

**Description**
The error status.

**Property Value**
An int specifying the error. If there was no error, zero is returned.

**Remarks**
A non-zero return indicates a potential problem with the trainer.

**ErrorValue**
abstract public double ErrorValue {get; }

**Description**
The value of the error function minimized by the trainer.

**Property Value**
A double indicating the final value of the error function from the last training.

**Remarks**
Before training, NaN is returned.

**Method**

**Train**
abstract public void Train(Imsl.DataMining.Neural.Network network, double[,] xData, double[,] yData)

**Description**
Trains the neural network using supplied training patterns.

**Parameters**
- network – A Network object, which is the Network to be trained.
- xData – A double matrix containing the input training patterns.
- yData – A double matrix containing the output training patterns.
Remarks
The number of columns in xData must equal the number of nodes in the InputLayer. Each row of xData contains a training pattern.
The number of columns in yData must equal the number of Perceptrons in the OutputLayer. Each row of yData contains a training pattern.

QuasiNewtonTrainer Class

public class Imsl.DataMining.Neural.QuasiNewtonTrainer : Imsl.DataMining.Neural.ITrainer, ICloneable
Trains a Network using the quasi-Newton method, MinUnconMultiVar.

See Also
Imsl.Math.MinUnconMultiVar (p. 344)

Field
SUM_OF_SQUARES
public Imsl.DataMining.Neural.QuasiNewtonTrainer.IError SUM_OF_SQUARES

Description
Compute the sum of squares error.

Remarks
The sum of squares error term is \( e(y, \hat{y}) = (y - \hat{y})^2 / 2 \).
This is the default IError object used by QuasiNewtonTrainer.

Properties

EpochNumber
virtual protected internal int EpochNumber {set; }

Description
The epoch number for the trainer.
Property Value
An int array containing the epoch number.

Error
virtual public Imsl.DataMining.Neural.QuasiNewtonTrainer.IError Error {get; set; }

Description
The error function used by the trainer.

Property Value
An IError object representing the error function used by the trainer.

ErrorGradient
virtual public double[] ErrorGradient {get; }

Description
The value of the gradient of the error function with respect to the Weights (p. 1666).

Property Value
A double array whose length is equal to the number of network Weights, containing the value of the gradient of the error function with respect to the Weights.

Remarks
Before training, null is returned.

ErrorStatus
virtual public int ErrorStatus {get; }

Description
The error status from the trainer.

Property Value
An int representing the error status from the trainer.

Remarks
Zero indicates that no errors were encountered during training. Any non-zero value indicates that some error condition arose during training. In many cases the trainer is able to recover from these conditions and produce a well-trained network.
<table>
<thead>
<tr>
<th>Status</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error occurred during training.</td>
</tr>
<tr>
<td>1</td>
<td>The last global step failed to locate a lower point than the current error value. The current solution may be an approximate solution and no more accuracy is possible, or the step tolerance may be too large.</td>
</tr>
<tr>
<td>2</td>
<td>Relative function convergence; both the actual and predicted relative reductions in the error function are less than or equal to the relative function convergence tolerance.</td>
</tr>
<tr>
<td>3</td>
<td>Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or the step tolerance is too big.</td>
</tr>
<tr>
<td>4</td>
<td>Optimizer threw a <code>FalseConvergenceException</code>.</td>
</tr>
<tr>
<td>5</td>
<td>Optimizer threw a <code>MaxIterationsException</code>.</td>
</tr>
<tr>
<td>6</td>
<td>Optimizer threw a <code>UnboundedBelowException</code>.</td>
</tr>
</tbody>
</table>


**ErrorValue**

```csharp
virtual public double ErrorValue {get; }
```

**Description**
The final value of the error function.

**Property Value**
A double representing the final value of the error function from the last training.

**Remarks**
Before training, NaN is returned.

**GradientTolerance**

```csharp
virtual public double GradientTolerance {get; set; }
```

**Description**
The gradient tolerance.

**Property Value**
A double specifying the gradient tolerance.

**Remarks**
By default, `GradientTolerance` equals the cube root of machine precision.

See Also: Imsl.Math.MinUnconMultiVar.GradientTolerance (p. 346)

**MaximumStepsize**

```csharp
virtual public double MaximumStepsize {get; set; }
```
Description
The maximum step size.

Property Value
A nonnegative double value specifying the maximum allowable step size in the optimizer.

Remarks
The value of MaximumStepsize will be equal to -999.0 if the default value is to be used and the Train method has not been called.

See Also: (p. 347)

MaximumTrainingIterations
virtual public int MaximumTrainingIterations {get; set; }

Description
The maximum number of iterations to use in a training.

Property Value
An int representing the maximum number of training iterations.

Remarks
By default, MaximumTrainingIterations = 100.

See Also: (p. 347)

NumberOfProcessors
public int NumberOfProcessors {get; set; }

Description
Perform the parallel calculations with the maximum possible number of processors set to NumberOfProcessors.

Property Value
An int indicating the maximum possible number of processors to use.

Remarks
By default, NumberOfProcessors = Environment.ProcessorCount. If NumberOfProcessors is set to a number less than 1 or greater than Environment.ProcessorCount, Environment.ProcessorCount will be used internally. This property has no effect if the application is used with a scalar version of the IMSL C# Numerical Library.

ParallelMode
virtual protected internal System.Collections.ArrayList[] ParallelMode {set; }

Description
The trainer to be used in multi-threaded EpochTainer.
**Property Value**
An ArrayList array containing the log records.

**StepTolerance**
`virtual public double StepTolerance {get; set; }

**Description**
The scaled step tolerance.

**Property Value**
A double which is the step tolerance.

**Remarks**
The second stopping criterion for MinUnconMultiVar (p. 344), the optimizer used by this ITrainer, is that the scaled distance between the last two steps be less than the step tolerance.
by default, StepTolerance = 3.66685e-11.
See Also: Imsl.Math.MinUnconMultiVar.StepTolerance (p. 348)

**TrainingIterations**
`virtual public int TrainingIterations {get; }

**Description**
The number of iterations used during training.

**Property Value**
An int representing the number of iterations used during training.
See Also: (p. 347)

**UseBackPropagation**
`virtual public bool UseBackPropagation {get; set; }

**Description**
Specify the use of the back propagation algorithm for gradient calculations during network training.

**Property Value**
A bool specifying whether or not back propagation is used for gradient calculations. By default, UseBackPropagation = true.

**Remarks**
By default, the quasi-newton algorithm optimizes the network using numerical gradients. This method directs the quasi-newton trainer to use the back propagation algorithm for gradient calculations during network training. Depending upon the data and network architecture, one approach is typically faster than the other, or is less sensitive to finding local network optima.
Constructor

QuasiNewtonTrainer  
public QuasiNewtonTrainer()

Description
Constructs a QuasiNewtonTrainer object.

Methods

Clone
virtual public object Clone()

Description
Clones a copy of the trainer.

GetError
virtual public Imsl.DataMining.Neural.QuasiNewtonTrainer.IError GetError()

Description
Returns the function used to compute the error to be minimized.

Returns
The IError object containing the function to be minimized.

SetError
virtual public void SetError(Imsl.DataMining.Neural.QuasiNewtonTrainer.IError error)

Description
Sets the function that computes the network error.

Parameter
  error – The IError object containing the function to be used to compute the network error.

Remarks
The default is to compute the sum of squares error, SUM_OF_SQUARES.

Train
virtual public void Train(Imsl.DataMining.Neural.Network network, double[,] xData, double[,] yData)

Description
Trains the neural network using supplied training patterns.
Parameters

- network – The Network to be trained.
- xData – An input double matrix containing training patterns.
- yData – An output double matrix containing output training patterns.

Remarks

The number of columns in xData must equal the number of Nodes in the InputLayer.
The number of columns in yData must equal the number of Perceptrons in the OutputLayer.
Each row of xData and yData contains a training pattern. The number of rows in these two arrays must be at least equal to the number of weights in the Network.

QuasiNewtonTrainer.IError Interface

public interface Imsl.DataMining.Neural.QuasiNewtonTrainer.IError

Error function to be minimized by trainer.
This trainer attempts to solve the problem

$$\min_w \sum_{i=0}^{n-1} e(y_i, \hat{y}_i)$$

where w are the weights, n is the number of training patterns, y_i is a training target output and \( \hat{y}_i \) is its forecast value.

This interface defines the function \( e(y, \hat{y}) \) and its derivative with respect to its computed value, \( de/d\hat{y} \).

Methods

Error

abstract public double Error(double[] computed, double[] expected)

Description

The contribution to the error from a single training output target. This is the function \( e(y_i, \hat{y}_i) \).

Parameters

- computed – A double representing the computed value.
- expected – A double representing the expected value.
Returns
A double representing the contribution to the error from a single training output target.

**ErrorGradient**
abstract public double[] ErrorGradient(double[] computed, double[] expected)

Description
The derivative of the error function with respect to the forecast output.

Parameters
- computed – A double representing the computed value.
- expected – A double representing the expected value.

Returns
A double representing the derivative of the error function with respect to the forecast output.

**LeastSquaresTrainer Class**

```csharp
public class Imsl.DataMining.Neural.LeastSquaresTrainer :
Imsl.DataMining.Neural.ITrainer

Trains a FeedForwardNetwork using a Levenberg-Marquardt algorithm for minimizing a sum of squares error.
```

**See Also**
Feed Forward Class Example 1, NonlinLeastSquares (p. 353)

**Properties**

**EpochNumber**
virtual protected internal int EpochNumber {set; }

Description
The epoch number for the trainer.
Property Value
An int array containing the epoch number.

ErrorGradient
virtual public double[] ErrorGradient {get; }

Description
The value of the gradient of the error function with respect to the Weights (p. 1666).

Property Value
A double array whose length is equal to the number of network weights, containing the value of the gradient of the error function with respect to the weights.

Remarks
Before training, null is returned.

ErrorStatus
virtual public int ErrorStatus {get; }

Description
The error status from the trainer.

Property Value
An int which contains the error status.

Remarks
Zero indicates that no errors were encountered during training. Any non-zero value indicates that some error condition arose during training.

In many cases the trainer is able to recover from these conditions and produce a well-trained network.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All convergence tests were met.</td>
</tr>
<tr>
<td>1</td>
<td>Scaled step tolerance was satisfied. The current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or StepTolerance is too big.</td>
</tr>
<tr>
<td>2</td>
<td>Scaled actual and predicted reductions in the function are less than or equal to the relative function convergence tolerance RelativeTolerance.</td>
</tr>
<tr>
<td>3</td>
<td>Iterates appear to be converging to a noncritical point. Incorrect gradient information, a discontinuous function, or stopping tolerances being too tight may be the cause.</td>
</tr>
<tr>
<td>4</td>
<td>Five consecutive steps with the maximum stepsize have been taken. Either the function is unbounded below, or has a finite asymptote in some direction, or the maximum stepsize is too small.</td>
</tr>
<tr>
<td>5</td>
<td>Too many iterations required.</td>
</tr>
</tbody>
</table>

ErrorValue
virtual public double ErrorValue {get; }
**Description**
The final value of the error function.

**Property Value**
A `double` containing the final value of the error function from the last training.

**Remarks**
Before training, `NaN` is returned.

---

**FalseConvergenceTolerance**
`virtual public double FalseConvergenceTolerance {get; set; }`

**Description**
The false convergence tolerance.

**Property Value**
A `double` specifying the false convergence tolerance.

**Remarks**
by default, `FalseConvergenceTolerance = 1.0e-14`.
See Also: `NonlinLeastSquares.FalseConvergenceTolerance (p. 355)`

---

**GradientTolerance**
`virtual public double GradientTolerance {get; set; }`

**Description**
The gradient tolerance.

**Property Value**
A `double` specifying the gradient tolerance.

**Remarks**
By default, `GradientTolerance = 2.0e-5`.
See Also: `NonlinLeastSquares.GradientTolerance (p. 355)`

---

**InitialTrustRegion**
`virtual public double InitialTrustRegion {get; set; }`

**Description**
The initial trust region.

**Property Value**
A `double` which specifies the initial trust region radius.

---

**Neural Nets LeastSquaresTrainer** 1677
Remarks
By default, InitialTrustRegion = -999.0, an unlimited trust region.
The value of InitialTrustRegion will be equal to -999.0 if the default value is to be used and the
Train method has not been called.
See Also: NonlinLeastSquares.InitialTrustRegion (p. 356)

MaximumStepsize
virtual public double MaximumStepsize {get; set; }

Description
The maximum step size.

Property Value
A nonnegative double value specifying the maximum allowable stepsize in the optimizer.

Remarks
By default, MaximumStepsize = \(10^{3}||w||_{2}\), where \(w\) are the values of the Weights (p. 1666) in the
network when training starts.
The value of MaximumStepsize will be equal to -999.0 if the default value is to be used and the Train
method has not been called.
See Also: NonlinLeastSquares.MaximumStepsize (p. 356)

MaximumTrainingIterations
virtual public int MaximumTrainingIterations {get; set; }

Description
The maximum number of iterations used by the nonlinear least squares solver.

Property Value
An int which specifies the maximum number of iterations to be used by the nonlinear least squares
solver.

Remarks
Its default value is 1000.
See Also: NonlinLeastSquares.RelativeTolerance (p. 357)

ParallelMode
virtual protected internal System.Collections.ArrayList[] ParallelMode {set; }

Description
The trainer to be used in multi-threaded EpochTainer.

Property Value
An ArrayList array containing the log records.

RelativeTolerance
virtual public double RelativeTolerance {get; set; }
Description
The relative tolerance.

Property Value
A double which specifies the relative error tolerance.

Remarks
It must be in the interval [0,1]. Its default value is 1.0e-20.
See Also: NonlinLeastSquares.RelativeTolerance (p. 357)

StepTolerance
virtual public double StepTolerance {get; set; }

Description
The step tolerance used to step between Weights (p. 1666).

Property Value
A double which specifies the scaled step tolerance to use when changing the weights.

Remarks
By default, StepTolerance = 1.0e-5.
See Also: NonlinLeastSquares.StepTolerance (p. 358)

Constructor

LeastSquaresTrainer
public LeastSquaresTrainer()

Description
Creates a LeastSquaresTrainer.

Method

Train
virtual public void Train(Imsl.DataMining.Neural.Network network, double[,] xData, double[,] yData)

Description
Trains the neural network using supplied training patterns.
Parameters

- network – The Network to be trained.
- xData – A double matrix which contains the input training patterns. The number of columns in xData must equal the number of Nodes in the InputLayer.
- yData – A double matrix which contains the output training patterns. The number of columns in yData must equal the number of Perceptrons in the OutputLayer.

Remarks

Each row of xData and yData contains a training pattern. These number of rows in two arrays must be equal.

---

**EpochTrainer Class**

```csharp
public class Imsl.DataMining.Neural.EpochTrainer : Imsl.DataMining.Neural.ITrainer

Two-stage training using randomly selected training patterns in stage I.

The EpochTrainer, is a meta-trainer that combines two trainers. The first trainer is used on a series of randomly selected subsets of the training patterns. For each subset, the Weights (p. 1788) are initialized to their initial values plus a random offset.

Stage II then refines the result found in stage I. The best result from the stage I trainings is used as the initial guess with the second trainer operating on the full set of training patterns. Stage II is optional, if the second trainer is null then the best stage I result is returned as the EpochTrainer’s result.

Properties

**EpochSize**

```csharp
virtual public int EpochSize {get; set; }
```

**Description**

The number of randomly selected training patterns in each stage I epoch.

**Property Value**

An int which contains the number of sample training patterns in each stage I epoch. The default value is the number of observations in the training data.

**ErrorGradient**

```csharp
virtual public double[] ErrorGradient {get; }
```
Description
The value of the gradient of the error function with respect to the Weights (p. 1788).

Property Value
A double array whose length is equal to the number of Network.Weights, containing the value of the gradient of the error function with respect to the weights.

Remarks
Before training, null is returned.

ErrorStatus
virtual public int ErrorStatus {get; }

Description
The training error status.

Property Value
An int containing the error status from stage II.

Remarks
If there is no stage II then the number of stage I epochs that returned a non-zero error status is returned.

ErrorValue
virtual public double ErrorValue {get; }

Description
The value of the error function.

Property Value
A double containing final value of the error function from the last training. Before training, NaN is returned.

NumberOfEpochs
virtual public int NumberOfEpochs {get; set; }

Description
The number of epochs used during stage I training.

Property Value
An int which contains the number of epochs used during stage I training. The default value is 10.

Random
virtual public Imsl.Stat.Random Random {get; set; }

Description
The random number generator used to perturb the stage I guesses.
Property Value
The Random object used to generate stage I perturbations.

Stage1Trainer
virtual protected internal Imsl.DataMining.Neural.ITrainer Stage1Trainer {get; }

Description
The stage 1 trainer.

Property Value
A Trainer containing the stage 1 trainer.

Stage2Trainer
virtual protected internal Imsl.DataMining.Neural.ITrainer Stage2Trainer {get; }

Description
The stage 1 trainer.

Property Value
A Trainer containing the stage 2 trainer.

Constructors

EpochTrainer
public EpochTrainer(Imsl.DataMining.Neural.ITrainer stage1Trainer)

Description
Creates a single stage EpochTrainer. Stage II training is bypassed.

Parameter
    stage1Trainer – The ITrainer used in stage I.

EpochTrainer
public EpochTrainer(Imsl.DataMining.Neural.ITrainer stage1Trainer,
Imsl.DataMining.Neural.ITrainer stage2Trainer)

Description
Creates a two-stage EpochTrainer.

Parameters
    stage1Trainer – The stage I ITrainer.
    stage2Trainer – The stage II ITrainer, or null if stage II is to be bypassed.
Methods

SetRandomSamples

Description
Sets the random number generators used to select random training patterns in stage I.

Parameters
randomA – A Random object which is the first random number generator.
randomB – A Random object which is the second random number generator, independent of randomA.

Remarks
The two random number generators should be independent.

Train
virtual public void Train(Imsl.DataMining.Neural.Network network, double[,] xData, double[,] yData)

Description
Trains the neural network using supplied training patterns.

Parameters
network – The Network to be trained.
xData – A double matrix specifying the input training patterns. The number of columns in xData must equal the number of Nodes in the InputLayer.
yData – A double containing the output training patterns. The number of columns in yData must equal the number of Perceptrons in the OutputLayer.

Remarks
Each row of xData and yData contains a training pattern. These number of rows in two arrays must be equal.

BinaryClassification Class

public class Imsl.DataMining.Neural.BinaryClassification
Classifies patterns into two classes.

Uses a FeedForwardNetwork to solve binary classification problems. In these problems, the target output for the network is the probability that the pattern falls into one of two classes. The first class,
\( P(C_1) \), is usually equal to one and the second class, \( P(C_2) \) equal to zero. These probabilities are then used to assign patterns to one of the two classes. Typical applications include determining whether a credit applicant is a good or bad credit risk, and determining whether a person should or should not receive a particular treatment based upon their physical, clinical and laboratory information. This class signals that network training will minimize the binary cross-entropy error, and that network output is the probability that the pattern belongs to the first class, \( P(C_1) \). Which is calculated by applying the logistic activation function to the potential of the single output. The probability for the second class is calculated by \( P(C_2) = 1 - P(C_1) \).

**Properties**

**Error**

virtual public Imsl.DataMining.Neural.QuasiNewtonTrainer.IError Error {get; }

**Description**

Returns the error function for use by QuasiNewtonTrainer for training a binary classification network.

**Property Value**

An implementation of the binary-entropy error function.

**Network**

virtual public Imsl.DataMining.Neural.Network Network {get; }

**Description**

The network being used for classification.

**Property Value**

The network set by the constructor.

**Constructor**

**BinaryClassification**

public BinaryClassification(Imsl.DataMining.Neural.Network network)

**Description**

Creates a binary classifier.

**Parameter**

network – Is the neural network used for classification. Its OutputPerceptron should use the logistic activation function, Imsl.DataMining.Neural.Activation.Logistic.
Methods

**ComputeStatistics**

```csharp
virtual public double[] ComputeStatistics(double[,] xData, int[] yData)
```

**Description**
Computes the classification error statistics for the supplied network patterns and their associated classifications.

**Parameters**
- `xData` – A `double` matrix specifying the input training patterns. The number of columns in `xData` must equal the number of Nodes in the InputLayer.
- `yData` – An `int` array containing the output classification patterns. The values in `yData` must be 0 or 1.

**Returns**
A two-element `double` array containing the binary cross-entropy error and the classification error rate.

**Remarks**
The first element returned is the binary cross-entropy error; the second is the classification error rate. The classification error rate is calculated by comparing the estimated classification probabilities to the target classifications. If the estimated probability for the target class is less than 0.5, then this is tallied as a classification error.

**PredictedClass**

```csharp
virtual public int PredictedClass(double[] x)
```

**Description**
Calculates the classification probabilities for the input pattern `x`, and returns either 0 or 1 identifying the class with the highest probability.

**Parameter**
- `x` – The `double` array containing the network input patterns to classify. The length of `x` should be equal to the number of inputs in the network.

**Returns**
The classification predicted by the trained network for `x`. This will be either 0 or 1.

**Remarks**
This method is used to classify patterns into one of the two target classes based upon the pattern’s values. The predicted classification is the class with the largest probability, i.e. greater than 0.5.

**Probabilities**

```csharp
virtual public double[] Probabilities(double[] x)
```

**Description**
Returns classification probabilities for the input pattern `x`. 

---

**Neural Nets**

**BinaryClassification** • 1685
Parameter

- **x** – A double array containing the network input pattern to classify. The length of x must equal the number of nodes in the input layer.

Returns

The probability of x being in class $C_1$, followed by the probability of x being in class $C_2$.

Remarks

Calculates the two probabilities for the pattern supplied: $P(C_1)$ and $P(C_2)$. The probability that the pattern belongs to the first class, $P(C_1)$, is estimated using the logistic function of the OutputPerceptron's potential. The probability for the second class is calculated as $P(C_2) = 1 - P(C_1)$. The predicted classification is the class with the largest probability, i.e. greater than 0.5.

Train

virtual public void Train(Imsl.DataMining.Neural.ITrainer trainer, double[,] xData, int[] yData)

Description

Trains the classification neural network using supplied trainer and patterns.

Parameters

- **trainer** – A Trainer object, which is used to train the network. The error function in any QuasiNewton trainer included in trainer should be set to the error function from this class using the Error method provided by this class.
- **xData** – A double matrix containing the input training patterns. The number of columns in xData must equal the number of nodes in the input layer. Each row of xData contains a training pattern.
- **yData** – An int array containing the output classification values. These values must be 0 or 1.

Example 1: Binary Classification

This example trains a 3-layer network using 48 training patterns from four nominal input attributes. The first two nominal attributes have two classifications. The third and fourth nominal attributes have three and four classifications respectively. All four attributes are encoded using binary encoding. This results in eleven binary network input columns. The output class is 1 if the first two nominal attributes sum to 1, and 0 otherwise.

The structure of the network consists of eleven input nodes and three layers, with three perceptrons in the first hidden layer, two perceptrons in the second hidden layer, and one perceptron in the output layer.

There are a total of 47 weights in this network, including the six bias weights. The linear activation function is used for both hidden layers. Since the target output is binary classification the logistic activation function is used in the output layer. Training is conducted using the quasi-newton trainer with the binary-entropy error function provided by the BinaryClassification class.

using System;
using Imsl.DataMining.Neural;
using Random = Imsl.Stat.Random;
using System.Runtime.Serialization;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;

//*****************************************************************************
// Two Layer Feed-Forward Network with 11 inputs: 4 nominal with 2,2,3,4
categories, encoded using binary encoding, and 1 output target (class).
//
// new classification training_ex1.c
//*****************************************************************************

[Serializable]
public class BinaryClassificationEx1
{

    // Network Settings
    private static int nObs = 48; // number of training patterns
    private static int nInputs = 11; // four nominal with 2,2,3,4 categories
    private static int nCategorical = 11; // three categorical attributes
    private static int nOutputs = 1; // one continuous output (nclasses=2)
    private static int nPerceptrons1 = 3; // perceptrons in 1st hidden layer
    private static int nPerceptrons2 = 2; // perceptrons in 2nd hidden layer

    private static IActivation hiddenLayerActivation =
        Imsl.DataMining.Neural.Activation.Linear;
    private static IActivation outputLayerActivation =
        Imsl.DataMining.Neural.Activation.Logistic;

    /* 2 classifications */
    private static int[] x1 = new int[]{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2};

    /* 2 classifications */
    private static int[] x2 = new int[]{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2};

    /* 3 classifications */
    private static int[] x3 = new int[]{1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 1,
        1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3,
        1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3};

    /* 4 classifications */
    private static int[] x4 = new int[]{1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1,
        2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1,
        2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1};

    // **********************************************************************
    // MAIN
    // **********************************************************************

    public static void Main(System.String[] args)
    {
        double[,] xData; // Input Attributes for Trainer
    
    Neural Nets BinaryClassification ● 1687
int[] yData; // Output Attributes for Trainer
int i, j; // array indicies

// Binary encode 4 categorical variables.
// Var x1 contains 2 classes
// Var x2 contains 2 classes
// Var x3 contains 3 classes
// Var x4 contains 4 classes
// ******************************************************************
int[,] z1;
int[,] z2;
int[,] z3;
int[,] z4;
UnsupervisedNominalFilter filter = new UnsupervisedNominalFilter(2);
z1 = filter.Encode(x1);
z2 = filter.Encode(x2);
filter = new UnsupervisedNominalFilter(3);
z3 = filter.Encode(x3);
filter = new UnsupervisedNominalFilter(4);
z4 = filter.Encode(x4);

/* Concatenate binary encoded z's */
xData = new double[nObs,nInputs];
yData = new int[nObs];
for (i = 0; i < (nObs); i++)
{
    for (j = 0; j < nCategorical; j++)
    {
        xData[i,j] = 0;
        if (j < 2)
            xData[i,j] = (double) z1[i,j];
        if (j > 1 && j < 4)
            xData[i,j] = (double) z2[i,j - 2];
        if (j > 3 && j < 7)
            xData[i,j] = (double) z3[i,j - 4];
        if (j > 6)
            xData[i,j] = (double) z4[i,j - 7];
    }
    yData[i] = ((x1[i] + x2[i] == 2)?1:0);
}

// **********************************************************************
// CREATE FEEDFORWARD NETWORK
// **********************************************************************
long t0 = (System.DateTime.Now.Ticks - 621355968000000000) / 10000;

FeedForwardNetwork network = new FeedForwardNetwork();
network.InputLayer.CreateInputs(nInputs);
network.CreateHiddenLayer().CreatePerceptrons(nPerceptrons1);
network.CreateHiddenLayer().CreatePerceptrons(nPerceptrons2);
network.OutputLayer.CreatePerceptrons(nOutputs);

BinaryClassification classification = new BinaryClassification(network);
network.LinkAll();
System.Random r = new System.Random(123457);
network.SetRandomWeights(xData, r);
Perceptron[] perceptrons = network.Perceptrons;
for (i = 0; i < perceptrons.Length - 1; i++)
{
    perceptrons[i].Activation = hiddenLayerActivation;
}
perceptrons[perceptrons.Length - 1].Activation = outputLayerActivation;

// TRAIN NETWORK USING QUASI-NEWTON TRAINER
// **********************************************************************
QuasiNewtonTrainer trainer = new QuasiNewtonTrainer();
trainer.Error = classification.Error;
trainer.MaximumTrainingIterations = 1000;
trainer.MaximumStepsize = 3.0;
trainer.GradientTolerance = 1.0e-20;
trainer.StepTolerance = 1.0e-20;
classification.Train(trainer, xData, yData);

// DISPLAY TRAINING STATISTICS
// **********************************************************************
double[] stats = classification.ComputeStatistics(xData, yData);
System.Console.Out.WriteLine("***********************************************");
System.Console.Out.WriteLine("-- Cross-entropy error: "+(float)stats[0]);
System.Console.Out.WriteLine("-- Classification error rate: "+(float)stats[1]);
System.Console.Out.WriteLine("***********************************************");
System.Console.Out.WriteLine(" ");

// OBTAIN AND DISPLAY NETWORK WEIGHTS AND GRADIENTS
// **********************************************************************
double[] weight = network.Weights;
double[] gradient = trainer.ErrorGradient;
double[,] wg = new double[weight.Length, 2];
for (i = 0; i < weight.Length; i++)
{
    wg[i, 0] = weight[i];
    wg[i, 1] = gradient[i];
}
PrintMatrixFormat pmf = new PrintMatrixFormat();
pmf.SetColumnLabels(new System.String[] {"Weights", "Gradients"});
new PrintMatrix().Print(pmf, wg);

// forecast the network
// **********************************************************************
double[,] report = new double[nObs, 6];
for (i = 0; i < nObs; i++)
{
{ report[i,0] = x1[i];
    report[i,1] = x2[i];
    report[i,2] = x3[i];
    report[i,3] = x4[i];
    report[i,4] = yData[i];
    double[] tmp = new double[xData.GetLength(1)];
    for (j=0; j<xData.GetLength(1); j++)
        tmp[j] = xData[i,j];
    report[i,5] = classification.PredictedClass(tmp);
}

pmf = new PrintMatrixFormat();
pmf.SetColumnLabels( new System.String[]{"X1", "X2", "X3", "X4",
                      "Expected", "Predicted"});
new PrintMatrix("Forecast").Print(pmf, report);

// **********************************************************************
// DISPLAY CLASSIFICATION STATISTICS
// **********************************************************************
double[] statsClass = classification.ComputeStatistics(xData, yData);
// Display Network Errors
                     (float)statsClass[0]);
System.Console.Out.WriteLine("--> Classification Error: " +
                     (float)statsClass[1]);
System.Console.Out.WriteLine("\n");

long t1 = (System.DateTime.Now.Ticks - 621355968000000000) / 10000;
double time = t1 - t0;
time = time / 1000;
System.Console.Out.WriteLine("**********Time: " + time);
System.Console.Out.WriteLine("trainer.getErrorValue = " +
                    trainer.ErrorValue);
}

Output
*******************************
--> Cross-entropy error: 4.720552E-10
--> Classification error rate: 0
*******************************

<table>
<thead>
<tr>
<th>Weights</th>
<th>Gradients</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.21627824426795</td>
</tr>
<tr>
<td>1</td>
<td>-7.10104582036496</td>
</tr>
<tr>
<td>2</td>
<td>-4.8639364224764</td>
</tr>
<tr>
<td>3</td>
<td>-2.60725959847449</td>
</tr>
<tr>
<td>4</td>
<td>4.29051046748516</td>
</tr>
</tbody>
</table>

1690 • BinaryClassification

IMSL C# Numerical Library
<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
<th>Expected</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Neural Nets  
BinaryClassification  
1691
Example 2: Binary Classification Network

This example uses a database of a complete set of possible board configurations at the end of tic-tac-toe games, where “x” is assumed to have played first. The target concept is “win for x” (i.e., true when “x” has one of 8 possible ways to create a “three-in-a-row”).

There are nine nominal input attributes for each square on the tic-tac-toe board and are encoded such that 0=player x has taken, 1=player o has taken, 2=blank.

1692 • BinaryClassification  IMSL C# Numerical Library
**Input attributes**

1. top-left-square: \{x,o,b\}
2. top-middle-square: \{x,o,b\}
3. top-right-square: \{x,o,b\}
4. middle-left-square: \{x,o,b\}
5. middle-middle-square: \{x,o,b\}
6. middle-right-square: \{x,o,b\}
7. bottom-left-square: \{x,o,b\}
8. bottom-middle-square: \{x,o,b\}
9. bottom-right-square: \{x,o,b\}

The predicted attribute is a win or lose at tic-tac-toe. For this example the first 626 observations are a win and the next 332 are loss.

The structure of the network consists of 27 input nodes and three layers, with five perceptrons in the first hidden layer, three perceptrons in the second hidden layer, and one perceptron in the output layer.

There are a total of 162 weights in this network. The activations functions are logistic for all layers. Since the target output is binary classification the logistic activation function must be used in the output layer. Training is conducted using the quasi-newton trainer using the binary entropy error function provided by the **BinaryClassification** class.

```csharp
using System;
using Imsl.DataMining.Neural;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;
using Random = Imsl.Stat.Random;
//****************************************************************************
// Three Layer Feed-Forward Network with 4 inputs, all
// continuous, and 2 classification categories.
//
// new classification training_ex4.c
//
// Three Layer Feed-Forward Network with 4 inputs, all
// continuous, and 2 classification categories.
//
// This database encodes the complete set of possible board configurations
// at the end of tic-tac-toe games, where "x" is assumed to have played
// first. The target concept is "win for x" (i.e., true when "x" has one
// of 8 possible ways to create a "three-in-a-row")..
//
// Predicted attribute: win or loose at tic-tac-toe
// First 626 obs are positive (win) and the next 332 are negative (loss)
//```
// Input Attributes (10 categorical Attributes)
// Attribute Information: (0=player x has taken, 1=player o has taken, 2=blank)
//
// 1. top-left-square: {x,o,b}
// 2. top-middle-square: {x,o,b}
// 3. top-right-square: {x,o,b}
// 4. middle-left-square: {x,o,b}
// 5. middle-middle-square: {x,o,b}
// 6. middle-right-square: {x,o,b}
// 7. bottom-left-square: {x,o,b}
// 8. bottom-middle-square: {x,o,b}
// 9. bottom-right-square: {x,o,b}
// 10. Class: {positive,negative}

//*****************************************************************************

[Serializable]
public class BinaryClassificationEx2
{
    private static int nObs = 958; // number of training patterns
    private static int nInputs = 27; // 9 nominal coded as 0=x, 1=0, 2=blank
    private static int nOutputs = 1; // one continuous output (nClasses=2)
    private static int nPerceptrons1 = 5; // perceptrons in 1st hidden layer
    private static int nPerceptrons2 = 3; // perceptrons in 2nd hidden layer

    private static IActivation hiddenLayerActivation =
        Imsl.DataMining.Neural.Activation.Logistic;
    private static IActivation outputLayerActivation =
        Imsl.DataMining.Neural.Activation.Logistic;

    private static int[][] data = new int[][]{
        new int[]{0, 0, 0, 0, 1, 1, 0, 1, 0},
        new int[]{0, 0, 0, 0, 1, 1, 0, 0, 1},
        new int[]{0, 0, 0, 0, 1, 1, 0, 1, 1},
        new int[]{0, 0, 0, 0, 1, 1, 0, 2, 2},
        new int[]{0, 0, 0, 0, 1, 1, 1, 1, 2},
        new int[]{0, 0, 0, 0, 1, 1, 2, 1, 2},
        new int[]{0, 0, 0, 0, 1, 1, 2, 2, 1},
        new int[]{0, 0, 0, 0, 1, 1, 2, 2, 2},
        new int[]{0, 0, 0, 0, 1, 2, 1, 1, 2},
        new int[]{0, 0, 0, 0, 1, 2, 1, 2, 1},
        new int[]{0, 0, 0, 0, 1, 2, 2, 1, 1},
        new int[]{0, 0, 0, 0, 1, 2, 2, 1, 2},
    };

    // Serialization

new int[]{1, 1, 0, 1, 0, 2, 0, 2, 0}, new int[]{1, 1, 0, 1, 1, 0, 0, 0, 0},
new int[]{1, 1, 0, 1, 2, 0, 0, 2, 0}, new int[]{1, 1, 0, 1, 2, 0, 1, 0, 2},
new int[]{1, 1, 0, 1, 2, 2, 0, 0, 0}, new int[]{1, 1, 0, 2, 0, 0, 0, 1, 2},
new int[]{1, 1, 0, 2, 0, 0, 0, 2, 1}, new int[]{1, 1, 0, 2, 0, 0, 1, 2, 0},
new int[]{1, 1, 0, 2, 0, 0, 2, 1, 0}, new int[]{1, 1, 0, 2, 0, 1, 0, 0, 2},
new int[]{1, 1, 0, 2, 0, 1, 0, 2, 0}, new int[]{1, 1, 0, 2, 0, 2, 0, 0, 1},
new int[]{1, 1, 0, 2, 0, 2, 0, 1, 0}, new int[]{1, 1, 0, 2, 0, 2, 0, 2, 2},
new int[]{1, 1, 0, 2, 1, 0, 2, 0, 0}, new int[]{1, 1, 0, 2, 1, 0, 2, 0, 2},
new int[]{1, 1, 0, 2, 1, 2, 0, 0, 0}, new int[]{1, 1, 0, 2, 2, 0, 0, 1, 0},
new int[]{1, 1, 0, 2, 2, 0, 1, 0, 0}, new int[]{1, 1, 0, 2, 2, 0, 2, 0, 1},
new int[]{1, 1, 0, 2, 2, 0, 2, 1, 0}, new int[]{1, 1, 0, 2, 2, 1, 0, 0, 0},
new int[]{1, 1, 0, 2, 2, 1, 0, 1, 0}, new int[]{1, 1, 0, 2, 2, 1, 0, 2, 2},
new int[]{1, 1, 0, 2, 2, 1, 1, 0, 0}, new int[]{1, 1, 0, 2, 2, 1, 1, 0, 1},
new int[]{1, 1, 0, 2, 2, 2, 1, 0, 0}, new int[]{2, 0, 0, 0, 0, 0, 1, 2},
new int[]{2, 0, 0, 0, 0, 0, 2, 1}, new int[]{2, 0, 0, 0, 0, 1, 0, 2},
new int[]{2, 0, 0, 0, 1, 0, 0, 2}, new int[]{2, 0, 0, 0, 1, 0, 1, 0},
new int[]{2, 0, 0, 0, 1, 0, 2, 0}, new int[]{2, 0, 0, 0, 1, 0, 2, 1},
new int[]{2, 0, 0, 0, 1, 1, 0, 0}, new int[]{2, 0, 0, 0, 1, 1, 0, 1},
new int[]{2, 0, 0, 0, 1, 1, 0, 2}, new int[]{2, 0, 0, 0, 1, 1, 1, 0},
new int[]{2, 0, 0, 0, 1, 2, 1, 0}, new int[]{2, 0, 0, 0, 2, 1, 0, 0},
new int[]{2, 0, 0, 1, 0, 0, 0, 1}, new int[]{2, 0, 0, 1, 0, 0, 1, 0},
new int[]{2, 0, 0, 1, 0, 0, 1, 1}, new int[]{2, 0, 0, 1, 0, 1, 0, 0},
new int[]{2, 0, 0, 1, 0, 1, 0, 1}, new int[]{2, 0, 0, 1, 0, 2, 0, 0},
new int[]{2, 0, 0, 1, 0, 2, 0, 1}, new int[]{2, 0, 0, 1, 0, 2, 0, 2},
new int[]{2, 0, 0, 1, 0, 2, 0, 3}, new int[]{2, 0, 0, 1, 0, 2, 0, 4},
new int[]{2, 0, 0, 1, 0, 2, 0, 5}, new int[]{2, 0, 0, 1, 0, 2, 0, 6},
new int[]{2, 0, 0, 1, 0, 2, 0, 7}, new int[]{2, 0, 0, 1, 0, 2, 0, 8},
new int[]{2, 0, 0, 1, 0, 2, 0, 9}, new int[]{2, 0, 0, 1, 0, 2, 0, 10},
new int[]{2, 0, 0, 1, 0, 2, 0, 11}, new int[]{2, 0, 0, 1, 0, 2, 0, 12},
new int[]{2, 0, 0, 1, 0, 2, 0, 13}, new int[]{2, 0, 0, 1, 0, 2, 0, 14},
new int[]{2, 0, 0, 1, 0, 2, 0, 15}, new int[]{2, 0, 0, 1, 0, 2, 0, 16}
1698 • BinaryClassification • IMSL C# Numerical Library
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new

int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,

Neural Nets

0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,

1,
1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,

0,
1,
1,
1,
1,
1,
2,
2,
2,
1,
1,
1,
2,
2,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
0,
0,
0,
0,
1,
2,
2,
0,
0,
0,
1,
1,
2,
0,
1,
1,
1,
1,
1,
1,
2,
2,
2,
0,

0,
0,
0,
0,
0,
2,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
2,
0,
0,
0,
0,
1,
1,
2,
2,
0,
0,
0,
1,
1,
2,
0,
0,
0,
1,
0,
0,
2,
0,
0,
0,
0,
2,
2,
1,
0,
0,
0,
1,
1,
2,
0,
1,
2,
0,

1,
0,
1,
2,
2,
1,
1,
2,
1,
0,
1,
2,
1,
2,
0,
1,
2,
0,
0,
0,
1,
2,
0,
2,
0,
1,
0,
1,
2,
0,
1,
0,
0,
0,
0,
2,
2,
1,
2,
0,
0,
0,
1,
2,
1,
0,
0,
1,
2,
0,
0,
0,
1,
0,
0,
0,

1,
1,
0,
0,
2,
0,
1,
1,
0,
1,
1,
1,
1,
1,
1,
0,
0,
1,
0,
2,
0,
0,
0,
0,
1,
0,
1,
0,
0,
1,
0,
2,
0,
1,
2,
0,
0,
0,
0,
1,
1,
2,
0,
0,
0,
1,
1,
0,
0,
0,
2,
2,
0,
1,
1,
1,

0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
2,
1,
1,
1,
2,
1,
2,
1,
2,
0,
0,
0,
1,
1,
1,
0,
0,
1,
2,
2,
1,
0,
0,
0,
0,
0,
2,
2,
0,
0,
0,
1,
1,
1,
1,
1,
2,
1,
2,
2,
1,
0,

2},
2},
2},
1},
2},
0},
0},
2},
0},
1},
0},
2},
2},
1},
1},
2},
1},
0},
1},
0},
0},
0},
0},
0},
0},
0},
0},
0},
2},
0},
0},
0},
1},
0},
0},
0},
0},
0},
0},
1},
2},
1},
0},
0},
0},
0},
0},
0},
2},
0},
0},
0},
1},
0},
0},
1},

new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new

int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,

0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,

1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,

0,
1,
1,
1,
1,
2,
2,
2,
0,
1,
1,
1,
2,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
0,
0,
0,
0,
0,
1,
2,
0,
0,
0,
0,
1,
2,
0,
1,
1,
1,
1,
1,
1,
2,
2,
2,
0,
0,

0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
2,
2,
0,
0,
0,
0,
1,
2,
0,
0,
0,
0,
2,
2,
1,
0,
0,
0,
1,
1,
1,
0,
0,
0,
0,
0,
1,
2,
0,
0,
1,
0,
0,

2,
0,
1,
2,
2,
0,
1,
2,
1,
1,
1,
2,
1,
0,
0,
1,
0,
0,
0,
1,
2,
2,
0,
0,
0,
0,
1,
1,
2,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
0,
2,
1,
0,
1,
1,
2,
0,
0,
1,
2,
0,
0,
0,

1,
2,
2,
1,
0,
1,
2,
2,
1,
0,
2,
2,
2,
1,
2,
0,
1,
0,
1,
0,
0,
0,
2,
0,
2,
0,
0,
0,
0,
2,
1,
0,
1,
2,
2,
0,
0,
0,
0,
1,
2,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
0,
0,
2,
0,
1,

0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
2,
2,
1,
2,
0,
0,
2,
0,
1,
2,
1,
0,
2,
2,
2,
2,
1,
0,
0,
2,
0,
0,
0,
1,
1,
1,
0,
0,
0,
1,
1,
0,
2,
2,
0,
2,
1,
1,
1,
1,
1,

1},
1},
0},
0},
0},
1},
2},
1},
1},
1},
2},
1},
1},
2},
1},
1},
0},
2},
0},
2},
1},
2},
0},
0},
0},
1},
1},
2},
1},
0},
0},
2},
2},
1},
2},
0},
0},
0},
1},
0},
2},
0},
0},
0},
1},
1},
1},
2},
1},
0},
0},
2},
1},
0},
1},
0},

BinaryClassification • 1699


new int[]{2, 1, 0, 0, 1, 2, 2}, new int[]{2, 2, 1, 0, 0, 2, 1, 2},
new int[]{2, 2, 1, 1, 0, 1, 0, 0}, new int[]{2, 2, 1, 1, 0, 0, 0, 0},
new int[]{2, 2, 2, 0, 0, 1, 1, 0}, new int[]{2, 2, 2, 0, 0, 0, 2, 1},
new int[]{2, 2, 2, 1, 0, 1, 0, 0}, new int[]{2, 2, 2, 1, 0, 0, 0, 0},
new int[]{2, 2, 2, 2, 0, 0, 0, 0}, new int[]{2, 2, 2, 2, 0, 0, 0, 0},
new int[]{2, 2, 2, 2, 1, 0, 0, 0}, new int[]{2, 2, 2, 2, 1, 0, 0, 0},
new int[]{2, 2, 2, 2, 1, 1, 0, 0}, new int[]{2, 2, 2, 2, 1, 1, 0, 0},
new int[]{2, 2, 2, 2, 1, 2, 0, 0}, new int[]{2, 2, 2, 2, 1, 2, 0, 0},
new int[]{2, 2, 2, 2, 2, 0, 0, 0}, new int[]{2, 2, 2, 2, 2, 0, 0, 0},
new int[]{2, 2, 2, 2, 2, 1, 0, 0}, new int[]{2, 2, 2, 2, 2, 1, 0, 0},
new int[]{2, 2, 2, 2, 2, 2, 0, 0}, new int[]{2, 2, 2, 2, 2, 2, 0, 0},
new int[]{2, 2, 2, 2, 2, 2, 1, 0}, new int[]{2, 2, 2, 2, 2, 2, 1, 0},
new int[]{2, 2, 2, 2, 2, 2, 2, 0}, new int[]{2, 2, 2, 2, 2, 2, 2, 0},
new int[]{2, 2, 2, 2, 2, 2, 2, 1}, new int[]{2, 2, 2, 2, 2, 2, 2, 1},
new int[]{2, 2, 2, 2, 2, 2, 2, 2}, new int[]{2, 2, 2, 2, 2, 2, 2, 2},
new int[]{2, 2, 2, 2, 2, 2, 2, 3}, new int[]{2, 2, 2, 2, 2, 2, 2, 3},
new int[]{2, 2, 2, 2, 2, 2, 2, 4}, new int[]{2, 2, 2, 2, 2, 2, 2, 4},
new int[]{2, 2, 2, 2, 2, 2, 2, 5}, new int[]{2, 2, 2, 2, 2, 2, 2, 5},
new int[]{2, 2, 2, 2, 2, 2, 2, 6}, new int[]{2, 2, 2, 2, 2, 2, 2, 6},
new int[]{2, 2, 2, 2, 2, 2, 2, 7}, new int[]{2, 2, 2, 2, 2, 2, 2, 7},
new int[]{2, 2, 2, 2, 2, 2, 2, 8}, new int[]{2, 2, 2, 2, 2, 2, 2, 8},
new int[]{2, 2, 2, 2, 2, 2, 2, 9}, new int[]{2, 2, 2, 2, 2, 2, 2, 9},
new int[]{2, 2, 2, 2, 2, 2, 2, 10}, new int[]{2, 2, 2, 2, 2, 2, 2, 10},
new int[]{2, 2, 2, 2, 2, 2, 2, 11}, new int[]{2, 2, 2, 2, 2, 2, 2, 11},
new int[]{2, 2, 2, 2, 2, 2, 2, 12}, new int[]{2, 2, 2, 2, 2, 2, 2, 12},
new int[]{2, 2, 2, 2, 2, 2, 2, 13}, new int[]{2, 2, 2, 2, 2, 2, 2, 13},
new int[]{2, 2, 2, 2, 2, 2, 2, 14}, new int[]{2, 2, 2, 2, 2, 2, 2, 14},
new int[]{2, 2, 2, 2, 2, 2, 2, 15}, new int[]{2, 2, 2, 2, 2, 2, 2, 15},
new int[]{2, 2, 2, 2, 2, 2, 2, 16}, new int[]{2, 2, 2, 2, 2, 2, 2, 16},
new int[]{2, 2, 2, 2, 2, 2, 2, 17}, new int[]{2, 2, 2, 2, 2, 2, 2, 17},
new int[]{2, 2, 2, 2, 2, 2, 2, 18}, new int[]{2, 2, 2, 2, 2, 2, 2, 18},
new int[]{2, 2, 2, 2, 2, 2, 2, 19}, new int[]{2, 2, 2, 2, 2, 2, 2, 19},
new int[]{2, 2, 2, 2, 2, 2, 2, 20}, new int[]{2, 2, 2, 2, 2, 2, 2, 20},
new int[]{2, 2, 2, 2, 2, 2, 2, 21}, new int[]{2, 2, 2, 2, 2, 2, 2, 21},
new int[]{2, 2, 2, 2, 2, 2, 2, 22}, new int[]{2, 2, 2, 2, 2, 2, 2, 22},
new int[]{2, 2, 2, 2, 2, 2, 2, 23}, new int[]{2, 2, 2, 2, 2, 2, 2, 23},
new int[]{2, 2, 2, 2, 2, 2, 2, 24}, new int[]{2, 2, 2, 2, 2, 2, 2, 24},
new int[]{2, 2, 2, 2, 2, 2, 2, 25}, new int[]{2, 2, 2, 2, 2, 2, 2, 25},
new int[]{2, 2, 2, 2, 2, 2, 2, 26}, new int[]{2, 2, 2, 2, 2, 2, 2, 26},
new int[]{2, 2, 2, 2, 2, 2, 2, 27}, new int[]{2, 2, 2, 2, 2, 2, 2, 27},
new int[]{2, 2, 2, 2, 2, 2, 2, 28}, new int[]{2, 2, 2, 2, 2, 2, 2, 28},
new int[]{2, 2, 2, 2, 2, 2, 2, 29}, new int[]{2, 2, 2, 2, 2, 2, 2, 29},
new int[]{2, 2, 2, 2, 2, 2, 2, 30}, new int[]{2, 2, 2, 2, 2, 2, 2, 30},
new int[]{2, 2, 2, 2, 2, 2, 2, 31}, new int[]{2, 2, 2, 2, 2, 2, 2, 31},
new int[]{2, 2, 2, 2, 2, 2, 2, 32}, new int[]{2, 2, 2, 2, 2, 2, 2, 32},
new int[]{2, 2, 2, 2, 2, 2, 2, 33}, new int[]{2, 2, 2, 2, 2, 2, 2, 33},
new int[]{2, 2, 2, 2, 2, 2, 2, 34}, new int[]{2, 2, 2, 2, 2, 2, 2, 34},
new int[]{2, 2, 2, 2, 2, 2, 2, 35}, new int[]{2, 2, 2, 2, 2, 2, 2, 35},
new int[]{2, 2, 2, 2, 2, 2, 2, 36}, new int[]{2, 2, 2, 2, 2, 2, 2, 36},
new int[]{2, 2, 2, 2, 2, 2, 2, 37}, new int[]{2, 2, 2, 2, 2, 2, 2, 37},
new int[]{2, 2, 2, 2, 2, 2, 2, 38}, new int[]{2, 2, 2, 2, 2, 2, 2, 38},
new int[]{2, 2, 2, 2, 2, 2, 2, 39}, new int[]{2, 2, 2, 2, 2, 2, 2, 39},
new int[]{0, 2, 1, 0, 1, 2, 1, 0, 2}, new int[]{0, 2, 2, 1, 0, 2, 1, 0, 0},
new int[]{0, 2, 2, 1, 1, 0, 2, 0, 1}, new int[]{0, 2, 2, 2, 0, 0, 2, 1, 1},
new int[]{0, 2, 2, 2, 1, 0, 2, 0, 2}, new int[]{0, 2, 2, 2, 2, 0, 0, 2, 0},
new int[]{1, 0, 0, 0, 0, 0, 0, 0, 0}, new int[]{1, 0, 0, 0, 0, 0, 0, 1, 0},
new int[]{1, 0, 0, 0, 0, 0, 1, 0, 0}, new int[]{1, 0, 0, 0, 0, 0, 1, 1, 0},
new int[]{1, 0, 0, 0, 0, 1, 0, 0, 0}, new int[]{1, 0, 0, 0, 0, 1, 0, 0, 1},
new int[]{1, 0, 0, 0, 0, 1, 0, 1, 0}, new int[]{1, 0, 0, 0, 0, 1, 1, 0, 0},
new int[]{1, 0, 0, 0, 1, 0, 0, 0, 0}, new int[]{1, 0, 0, 0, 1, 0, 0, 1, 0}
Neural Nets BinaryClassification
1701
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new

int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{0,
int[]{0,

1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
2,
0,
1,

1,
1,
1,
1,
1,
1,
2,
2,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
2,
2,
2,
2,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
0,
0,
0,
1,
1,
2,
2,
0,
0,
0,
1,
1,
1,
1,
1,
0,

2,
2,
2,
2,
2,
2,
0,
1,
0,
0,
0,
1,
1,
1,
1,
2,
0,
0,
0,
0,
1,
1,
0,
0,
1,
1,
1,
2,
0,
0,
0,
0,
0,
0,
2,
2,
2,
0,
1,
2,
0,
0,
2,
0,
0,
0,
2,
0,
1,
2,
0,
0,
0,
2,
1,
0,

1702 • BinaryClassification

0,
0,
0,
0,
0,
2,
1,
0,
1,
1,
1,
0,
0,
1,
2,
1,
0,
1,
1,
1,
0,
2,
0,
2,
1,
1,
1,
0,
0,
1,
1,
1,
1,
2,
0,
1,
2,
2,
1,
0,
1,
1,
1,
0,
1,
1,
1,
2,
1,
0,
0,
1,
2,
1,
1,
0,

0,
0,
0,
2,
2,
0,
0,
0,
0,
1,
2,
0,
2,
0,
2,
0,
1,
0,
0,
2,
0,
0,
1,
2,
1,
1,
1,
2,
1,
0,
0,
1,
2,
1,
1,
0,
1,
0,
1,
0,
0,
2,
0,
1,
0,
0,
0,
0,
1,
0,
1,
0,
1,
0,
0,
1,

0,
0,
2,
0,
2,
0,
0,
1,
1,
0,
2,
1,
1,
0,
1,
2,
0,
1,
0,
0,
1,
1,
1,
1,
0,
0,
2,
1,
0,
1,
1,
1,
1,
2,
0,
1,
0,
1,
0,
1,
2,
0,
0,
0,
1,
2,
0,
1,
0,
1,
2,
1,
0,
1,
0,
1,

0,
2,
0,
0,
0,
2,
0,
0,
0,
0,
0,
2,
0,
0,
0,
0,
0,
0,
2,
0,
2,
0,
1,
1,
0,
2,
0,
1,
1,
0,
2,
0,
2,
0,
2,
2,
0,
1,
2,
1,
1,
1,
1,
0,
0,
1,
1,
1,
2,
1,
0,
2,
0,
0,
0,
0,

1},
2},
2},
2},
0},
0},
1},
0},
1},
1},
1},
2},
2},
1},
0},
1},
1},
0},
1},
1},
0},
0},
1},
1},
1},
2},
2},
1},
1},
1},
2},
0},
0},
1},
1},
0},
1},
1},
0},
1},
2},
2},
2},
1},
0},
0},
0},
1},
0},
1},
1},
0},
1},
0},
1},
1},

new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new
new

int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{1,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{2,
int[]{0,
int[]{0,
int[]{0,

1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
2,
0,
0,
1,

1,
1,
1,
1,
1,
1,
2,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
2,
2,
2,
2,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
0,
0,
0,
0,
1,
2,
2,
0,
0,
0,
1,
1,
1,
1,
1,
1,
0,

2,
2,
2,
2,
2,
2,
0,
0,
0,
0,
1,
1,
1,
1,
2,
2,
0,
1,
0,
1,
1,
2,
0,
1,
1,
1,
1,
2,
0,
0,
0,
0,
0,
1,
2,
2,
0,
1,
1,
0,
0,
0,
2,
0,
0,
0,
0,
1,
1,
0,
0,
0,
2,
1,
1,
0,

0,
0,
0,
0,
2,
2,
1,
1,
1,
1,
0,
0,
0,
2,
1,
1,
1,
0,
1,
0,
0,
1,
1,
0,
1,
1,
1,
2,
0,
1,
1,
1,
2,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
0,
1,
1,
0,
0,
1,
1,

0,
0,
0,
2,
0,
0,
0,
0,
0,
2,
0,
1,
2,
0,
0,
2,
0,
0,
0,
0,
2,
0,
0,
0,
1,
1,
1,
0,
1,
0,
1,
2,
1,
0,
0,
2,
2,
1,
1,
0,
1,
2,
2,
0,
0,
2,
2,
1,
1,
1,
0,
2,
1,
0,
0,
0,

0,
1,
2,
0,
0,
2,
0,
0,
2,
0,
1,
1,
1,
1,
0,
0,
0,
1,
2,
1,
1,
0,
1,
1,
0,
1,
2,
1,
2,
1,
0,
1,
0,
1,
1,
1,
1,
0,
2,
0,
0,
2,
0,
0,
0,
0,
1,
0,
2,
0,
1,
1,
0,
0,
0,
1,

1,
0,
2,
2,
0,
0,
1,
1,
2,
2,
0,
0,
2,
0,
2,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
0,
2,
1,
2,
1,
0,
0,
2,
0,
0,
0,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
2,
0,
0,
0,
1,
1,
0,

0},
0},
0},
0},
2},
0},
0},
1},
1},
1},
1},
0},
0},
2},
1},
1},
1},
0},
1},
2},
0},
1},
1},
1},
0},
0},
0},
1},
1},
0},
1},
2},
1},
0},
2},
0},
1},
2},
0},
1},
0},
0},
0},
0},
2},
0},
1},
2},
0},
1},
2},
0},
1},
1},
0},
1},

IMSL C# Numerical Library


new int[]{0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1},
new int[]{0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0},
new int[]{1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0},
new int[]{1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1},
new int[]{1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0},
new int[]{1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1},
new int[]{1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0},
new int[]{1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1},
new int[]{1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0},
new int[]{1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1},
new int[]{1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0},
new int[]{1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1},
}

private static double[] weights = new double[]{
-0.00000000000000063401,
0.00000000000000055570,
0.00000000000000012769,
-0.52573653474162341000,
-0.43427498705107342000,
0.09146154769550232000,
0.000000000000000138130,
0.000000000000000518053,
0.00000000000000050631,
0.52573653474162607000,
-0.43427498705107603000,
-0.09146154769550940000,
-0.000000000000000057743,
-0.0000000000000000037314,
-0.0000000000000000023441,
0.52573653474162907000,
-0.43427498705107787000,
-0.09146154769551551000,
-0.00000000000000000405476,
0.00000000000000000339568,
0.0000000000000000053496,
-0.52573653474162763000,
-0.43427498705107587000,
0.09146154769551551000,
-0.00000000000000000116499,
0.00000000000000000111960,
0.000000000000000004466,
0.59184068444498500000,
-0.48617039139742850000,
-0.10564415450756450000,
0.33659639327260309000,
-0.28023189914604213000,
-0.05636504012656110000,
-0.33659639327260810000,
-0.0000000000000000000001,
0.00000000000000000312093,
0.33659639327260292000,
-0.28023189914604213000,
-0.05636504012656087800,
0.0000000000000000009480,
-0.0000000000000000067295,
-0.0000000000000000003901,
-0.33659639327260537000,
0.28023189914604435000,
0.05636504012656118300,
-0.000000000000000000284785,
-0.00000000000000000269180,
-0.33659639327260262000,
0.28023189914604330000,
0.05636504012656121800,
-0.59184068444493900000,
-0.48617039139733414000,
0.10564415450756090000,
0.0000000000000000098567,
-0.00000000000000000095474,
-0.0000000000000000021207,
-0.33659639327260698000,
0.28023189914604597000,
0.05636504012656142600,
-0.59184068444493720000,
0.48617039139733774000,
0.10564415450756450000,
0.33659639327260514000,
-0.28023189914604435000,
0.10564415450756450000,
0.33659639327260514000,
-0.28023189914604435000,
0.10564415450756450000,
0.33659639327260514000,
-0.28023189914604435000,
0.10564415450756450000,
0.33659639327260514000,
0.28023189914604597000,
0.05636504012656142600,
-0.59184068444493720000,
0.48617039139733774000,
0.10564415450756450000,
0.33659639327260514000,
-0.28023189914604435000,
0.10564415450756450000,
0.33659639327260514000,
-0.28023189914604435000,
0.10564415450756450000,
0.33659639327260514000,
0.28023189914604597000,
0.05636504012656142600,
-0.59184068444493720000,
0.48617039139733774000,
0.10564415450756450000,
0.33659639327260514000,
-0.28023189914604435000,
0.10564415450756450000,
0.33659639327260514000,
0.28023189914604597000,
0.05636504012656142600,
-0.59184068444493720000,
0.48617039139733774000,
0.10564415450756450000,
0.33659639327260514000,
public static void Main(System.String[] args)
{
    double[,] xData; // Input Attributes for Trainer
    int[] yData; // Output Attributes for Trainer
    int i, j; // array indicies
    int[,] z;

    // ************************************************************************
    // PREPROCESS TRAINING PATTERNS
    // ************************************************************************
    long t0 = (System.DateTime.Now.Ticks - 621355968000000000) / 10000;
    xData = new double[nObs,nInputs];
    yData = new int[nObs];

    /* Perform Binary Filtering. */
    for (i = 0; i < data.Length; i++)
    {
        for (j = 0; j < data[0].Length; j++)
        {
            data[i][j]++;
        }
    }
    int[] xx = new int[nObs];
    UnsupervisedNominalFilter filter = new UnsupervisedNominalFilter(3);
    for (i = 0; i < 9; i++)
    {
        // Copy each variable to a temp var
        for (j = 0; j < nObs; j++)
        {
            xx[j] = data[j][i];
        }
        // Perform binary filter on temp var
        z = filter.Encode(xx);
        // Copy binary encoded var to xData
        for (j = 0; j < nObs; j++)
        {
            for (int k = 0; k < 3; k++)
            {
                xData[j,k + (i * 3)] = (double) z[j,k];
            }
        }
    }
}
for (i = 0; i < nObs; i++)
{
    yData[i] = (i >= 62670?0:1);
}

// **********************************************************************
// CREATE FEEDFORWARD NETWORK
// **********************************************************************
FeedForwardNetwork network = new FeedForwardNetwork();
network.InputLayer.CreateInputs(nInputs);
network.CreateHiddenLayer().CreatePerceptrons(nPerceptrons1);
network.CreateHiddenLayer().CreatePerceptrons(nPerceptrons2);
network.OutputLayer.CreatePerceptrons(nOutputs);
network.LinkAll();
network.Weights = weights;
Perceptron[] perceptrons = network.Perceptrons;
for (i = 0; i < perceptrons.Length - 1; i++)
{
    perceptrons[i].Activation = hiddenLayerActivation;
}

// **********************************************************************
// SET OUTPUT LAYER ACTIVATION FUNCTION TO LOGISTIC FOR BINARY CLASSIFICATION
// **********************************************************************
perceptrons[perceptrons.Length - 1].Activation = outputLayerActivation;

BinaryClassification classification = new BinaryClassification(network);

QuasiNewtonTrainer stageITrainer = new QuasiNewtonTrainer();
QuasiNewtonTrainer stageIITrainer = new QuasiNewtonTrainer();
stageITrainer.SetError(classification.Error);
stageIITrainer.SetError(classification.Error);
stageITrainer.MaximumTrainingIterations = 8000;
stageIITrainer.MaximumTrainingIterations = 8000;
stageIITrainer.MaximumStepsize = 10.0;
stageIITrainer.MaximumStepsize = 10.0;
EpochTrainer trainer = new EpochTrainer(stageITrainer, stageIITrainer);

trainer.NumberOfEpochs = 20;
trainer.EpochSize = nObs;

trainer.Random = new Random(5555);
trainer.SetRandomSamples(new Random(5555), new Random(5555));
classification.Train(trainer, xData, yData);
System.Console.Out.WriteLine("trainer.getErrorValue = " +
    trainer.ErrorValue);
System.Console.Out.WriteLine("StageITrainer.getErrorValue = " +
    stageITrainer.ErrorValue);
System.Console.Out.WriteLine("StageIITrainer.getErrorValue = " +
    stageIITrainer.ErrorValue);

// **********************************************************************
// DISPLAY TRAINING STATISTICS
// **********************************************************************
double[] stats = classification.ComputeStatistics(xData, yData);
System.Console.Out.WriteLine("***********************************************");
System.Console.Out.WriteLine("--> Cross-entropy error: "+(float)stats[0]);
System.Console.Out.WriteLine("--> Classification error rate: "+(float)stats[1]);
System.Console.Out.WriteLine("***********************************************");

// **********************************************************************
// OBTAIN AND DISPLAY NETWORK WEIGHTS AND GRADIENTS
// **********************************************************************
double[] weight = network.Weights;
double[] gradient = trainer.ErrorGradient;
double[][] wg = new double[weight.Length][];
for (int i3 = 0; i3 < weight.Length; i3++)
{
    wg[i3] = new double[2];
}
for (i = 0; i < weight.Length; i++)
{
    wg[i][0] = weight[i];
    wg[i][1] = gradient[i];
}
PrintMatrixFormat pmf = new PrintMatrixFormat();

pmf.SetColumnLabels(new System.String[]("Weights", "Gradients");
new PrintMatrix().Print(pmf, wg);

// **********************************************************************
// forecast the network
// **********************************************************************
double[][] report = new double[nObs][];
for (int i4 = 0; i4 < nObs; i4++)
{
    report[i4] = new double[2];
}
for (i = 0; i < 50; i++)
{
    report[i][0] = yData[i];
    double[] tmp = new double[xData.GetLength(1)];
    for (j=0; j<xData.GetLength(1); j++)
        tmp[j] = xData[i,j];
    report[i][1] = classification.PredictedClass(tmp);
}

pmf = new PrintMatrixFormat();

pmf.SetColumnLabels( new System.String[]("Expected", "Predicted");
new PrintMatrix("Forecast").Print(pmf, report);

long t1 = (System.DateTime.Now.Ticks - 621355968000000000) / 10000;
double time = t1 - t0; //Math.max(small, (double)(t1-t0)/(double)iters);
time = time / 1000;
System.Console.Out.WriteLine("********Time: "+ time);
System.Console.Out.WriteLine("trainer.getErrorValue = " +

1706 • BinaryClassification

IMSL C# Numerical Library
trainer.ErrorValue);
System.Console.Out.WriteLine("StageITrainer.getErrorValue = " +
stageITrainer.ErrorValue);
System.Console.Out.WriteLine("StageIITrainer.getErrorValue = " +
stageIITrainer.ErrorValue);

Output

trainer.getErrorValue = 1.38629447264277
StageITrainer.getErrorValue = 341.896192465936
StageIITrainer.getErrorValue = 1.38629447264277

--> Cross-entropy error: 1.386294
--> Classification error rate: 0.002087683

Weights Gradients
0 -26.6097976573308 -1.76013093157232E-07
1 79.2470900942513 -3.9259607432252E-09
2 -33.5433683376609 -9.95507902332743E-09
3 2.21041535535485 6.47606304741283E-12
4 15.92279937086 7.92672053349369E-08
5 32.2932177476722 2.16769316126848E-07
6 -110.229528537971 1.60009076640888E-09
7 43.0115071508866 -9.56560614455885E-08
8 -6.72341801061297 -1.99042706347008E-08
9 3.82202932248916 -3.44338957093567E-09
10 -1.52763776833912 -4.67416809899007E-34
11 19.096315145894 1.065812685595036E-37
12 -6.20159756268499 -3.75471657439398E-37
13 3.30629260428251 1.77102057507537E-37
14 -26.3461658303238 -1.45338905665151E-42
15 -27.912063156244 -3.2315405378795E-08
16 36.3210304224857 1.55178803567368E-09
17 -40.570814070139 -1.00405066748808E-08
18 5.9418556798313 -2.10633388616164E-08
19 12.8231358566365 -8.1311631459708E-08
20 32.2113802373171 7.3071535553406E-08
21 -60.225886605208 -3.8776658012492E-09
22 42.834049469 -9.55706378121672E-08
23 0.253218038120759 1.1656390726295E-09
24 -10.419231245191 -1.39873355441285E-09
25 1.101320509594968 3.30048882191631E-27
26 12.19990492126 1.00942365101973E-27
27 3.156993388400009 -1.27487754292101E-36
28 -7.20259442259759 -4.97300992576816E-27
29 -9.9801831420775 2.66695912315823E-29
30 -26.338548742221 1.52366554789366E-07
31 65.165315620099 -2.59309440482311E-09
32 -40.2337421647594 -9.2886702769452E-08
33 12.7546549731031 1.16549218783095E-09
34 -36.991285381585 -3.4439063945978E-09
35 31.8719580928069 -1.11610331819749E-07

Neural Nets

BinaryClassification ● 1707
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>-75.4669369173807</td>
<td>2.67147071414864E-10</td>
</tr>
<tr>
<td>37</td>
<td>38.341769286413</td>
<td>-1.2724861919707E-08</td>
</tr>
<tr>
<td>38</td>
<td>-17.213954014748</td>
<td>-1.0633287621844E-08</td>
</tr>
<tr>
<td>39</td>
<td>18.4014307271567</td>
<td>-9.26720246364128E-08</td>
</tr>
<tr>
<td>40</td>
<td>-1.4308018594262</td>
<td>-3.8019373261246E-34</td>
</tr>
<tr>
<td>41</td>
<td>-0.26502137782834</td>
<td>8.4927031810026E-29</td>
</tr>
<tr>
<td>42</td>
<td>6.467823527918</td>
<td>-2.41991645073E-38</td>
</tr>
<tr>
<td>43</td>
<td>0.0087362956492</td>
<td>-4.97308613230902E-27</td>
</tr>
<tr>
<td>44</td>
<td>10.385823414027</td>
<td>3.279146651473E-30</td>
</tr>
<tr>
<td>45</td>
<td>-28.10247680355</td>
<td>-4.21249368544629E-07</td>
</tr>
<tr>
<td>46</td>
<td>62.810072678876</td>
<td>-2.2768589155942E-09</td>
</tr>
<tr>
<td>47</td>
<td>-23.9829618085034</td>
<td>-9.5665807793321E-08</td>
</tr>
<tr>
<td>48</td>
<td>-18.4761944978989</td>
<td>-1.98977918245327E-08</td>
</tr>
<tr>
<td>49</td>
<td>30.645136501639</td>
<td>-1.39871670591387E-09</td>
</tr>
<tr>
<td>50</td>
<td>32.433708210909</td>
<td>4.62005591614245E-07</td>
</tr>
<tr>
<td>51</td>
<td>-70.121815812784</td>
<td>-4.90140852568933E-11</td>
</tr>
<tr>
<td>52</td>
<td>56.790542920708</td>
<td>2.663105374416E-10</td>
</tr>
<tr>
<td>53</td>
<td>-53.7883610055096</td>
<td>-9.5570637836146E-08</td>
</tr>
<tr>
<td>54</td>
<td>27.984966690905</td>
<td>-1.98977923266468E-08</td>
</tr>
<tr>
<td>55</td>
<td>-54.490399816996</td>
<td>2.0446739347846E-09</td>
</tr>
<tr>
<td>56</td>
<td>37.051446511016</td>
<td>-2.47870004953234E-07</td>
</tr>
<tr>
<td>57</td>
<td>-63.9187059285026</td>
<td>-2.59222103056048E-09</td>
</tr>
<tr>
<td>58</td>
<td>57.7528833255766</td>
<td>-1.00450526327699E-08</td>
</tr>
<tr>
<td>59</td>
<td>-19.1914450829349</td>
<td>-2.4788663240107E-15</td>
</tr>
<tr>
<td>60</td>
<td>34.465888611249</td>
<td>-8.475526883951E-08</td>
</tr>
<tr>
<td>61</td>
<td>-13.109028974173</td>
<td>1.76210736423409E-24</td>
</tr>
<tr>
<td>62</td>
<td>5.49900711796567</td>
<td>1.00424771675945E-27</td>
</tr>
<tr>
<td>63</td>
<td>1.70369231662346</td>
<td>-5.32757842089505E-23</td>
</tr>
<tr>
<td>64</td>
<td>-9.0928992443768</td>
<td>-3.52004697402979E-26</td>
</tr>
<tr>
<td>65</td>
<td>12.2953916658469</td>
<td>2.3390441586071E-29</td>
</tr>
<tr>
<td>66</td>
<td>-27.48181929603831</td>
<td>4.43281192065308E-07</td>
</tr>
<tr>
<td>67</td>
<td>48.131658031404</td>
<td>-4.98371251655421E-11</td>
</tr>
<tr>
<td>68</td>
<td>-34.076114176618</td>
<td>-7.63854604777945E-09</td>
</tr>
<tr>
<td>69</td>
<td>-10.6698372282012</td>
<td>3.71928374478061E-10</td>
</tr>
<tr>
<td>70</td>
<td>20.7852963038748</td>
<td>-8.13118787157936E-08</td>
</tr>
<tr>
<td>71</td>
<td>32.5577122725433</td>
<td>-4.02524969096691E-07</td>
</tr>
<tr>
<td>72</td>
<td>-61.188735024785</td>
<td>-2.27603281650789E-09</td>
</tr>
<tr>
<td>73</td>
<td>34.8170651521053</td>
<td>-9.79725844211365E-08</td>
</tr>
<tr>
<td>74</td>
<td>6.63722464562796</td>
<td>-2.06972294883151E-08</td>
</tr>
<tr>
<td>75</td>
<td>16.5220845779096</td>
<td>-1.398716827893E-09</td>
</tr>
<tr>
<td>76</td>
<td>-0.160895088640464</td>
<td>3.30048890011806E-27</td>
</tr>
<tr>
<td>77</td>
<td>1.85827799132845</td>
<td>1.0935461724158E-27</td>
</tr>
<tr>
<td>78</td>
<td>3.78269514701215</td>
<td>-2.036035425952E-37</td>
</tr>
<tr>
<td>79</td>
<td>2.9797477464942</td>
<td>-7.72068070714235E-32</td>
</tr>
<tr>
<td>80</td>
<td>-3.6057967549916</td>
<td>2.45076692281205E-27</td>
</tr>
<tr>
<td>81</td>
<td>-27.659379760404</td>
<td>4.15232805134975E-07</td>
</tr>
<tr>
<td>82</td>
<td>72.3342855580823</td>
<td>-2.3258476757855E-09</td>
</tr>
</tbody>
</table>

1708 • BinaryClassification  IMSL C# Numerical Library
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>-24.7298826022964</td>
</tr>
<tr>
<td>93</td>
<td>5.24227794358502</td>
</tr>
<tr>
<td>94</td>
<td>-16.9304351865014</td>
</tr>
<tr>
<td>95</td>
<td>30.7407379285903</td>
</tr>
<tr>
<td>96</td>
<td>-71.9479456279238</td>
</tr>
<tr>
<td>97</td>
<td>45.8003226113044</td>
</tr>
<tr>
<td>98</td>
<td>-24.6378777857398</td>
</tr>
<tr>
<td>99</td>
<td>6.23728601645476</td>
</tr>
<tr>
<td>100</td>
<td>0.533861406588414</td>
</tr>
<tr>
<td>101</td>
<td>-11.5890280314343</td>
</tr>
<tr>
<td>102</td>
<td>-16.1548714189138</td>
</tr>
<tr>
<td>103</td>
<td>18.95357397183</td>
</tr>
<tr>
<td>104</td>
<td>3.56581181468853</td>
</tr>
<tr>
<td>105</td>
<td>-28.46593502998</td>
</tr>
<tr>
<td>106</td>
<td>21.3448850345718</td>
</tr>
<tr>
<td>107</td>
<td>-19.30027174882</td>
</tr>
<tr>
<td>108</td>
<td>-2.77292404194946</td>
</tr>
<tr>
<td>109</td>
<td>1.93277264147455</td>
</tr>
<tr>
<td>110</td>
<td>32.3382786494488</td>
</tr>
<tr>
<td>111</td>
<td>-49.952082003002</td>
</tr>
<tr>
<td>112</td>
<td>29.3493556080889</td>
</tr>
<tr>
<td>113</td>
<td>-5.47912940998204</td>
</tr>
<tr>
<td>114</td>
<td>-5.68496655040528</td>
</tr>
<tr>
<td>115</td>
<td>0.260715006887127</td>
</tr>
<tr>
<td>116</td>
<td>16.667759427184</td>
</tr>
<tr>
<td>117</td>
<td>-7.04065602302588</td>
</tr>
<tr>
<td>118</td>
<td>7.02117525437252</td>
</tr>
<tr>
<td>119</td>
<td>-3.11151101069225</td>
</tr>
<tr>
<td>120</td>
<td>-28.4825419589089</td>
</tr>
<tr>
<td>121</td>
<td>71.1271284091255</td>
</tr>
<tr>
<td>122</td>
<td>-32.416683013135</td>
</tr>
<tr>
<td>123</td>
<td>-1.70847780496737</td>
</tr>
<tr>
<td>124</td>
<td>-16.257204039972</td>
</tr>
<tr>
<td>125</td>
<td>30.338584173894</td>
</tr>
<tr>
<td>126</td>
<td>-87.9377967090049</td>
</tr>
<tr>
<td>127</td>
<td>50.246529583999</td>
</tr>
<tr>
<td>128</td>
<td>-10.09665405183</td>
</tr>
<tr>
<td>129</td>
<td>10.263349323559</td>
</tr>
<tr>
<td>130</td>
<td>1.96938482682724</td>
</tr>
<tr>
<td>131</td>
<td>5.34996503177684</td>
</tr>
<tr>
<td>132</td>
<td>-14.713205212991</td>
</tr>
<tr>
<td>133</td>
<td>11.716720259857</td>
</tr>
<tr>
<td>134</td>
<td>-2.011950080206</td>
</tr>
<tr>
<td>135</td>
<td>-165.787101153353</td>
</tr>
<tr>
<td>136</td>
<td>-153.126623517672</td>
</tr>
<tr>
<td>137</td>
<td>-21.440741371651</td>
</tr>
<tr>
<td>138</td>
<td>19.5580189313542</td>
</tr>
<tr>
<td>139</td>
<td>44.903949067269</td>
</tr>
<tr>
<td>140</td>
<td>-49.083352621682</td>
</tr>
<tr>
<td>141</td>
<td>-18.938169969533</td>
</tr>
<tr>
<td>142</td>
<td>-83.3719106138484</td>
</tr>
<tr>
<td>143</td>
<td>137.94476899566</td>
</tr>
<tr>
<td>144</td>
<td>39.031351432063</td>
</tr>
<tr>
<td>145</td>
<td>75.599341597783</td>
</tr>
<tr>
<td>146</td>
<td>195.16995384174</td>
</tr>
<tr>
<td>147</td>
<td>-83.019947688769</td>
</tr>
</tbody>
</table>
148  25.8184987265264  -1.62741417975303E-10
149  -97.6355018936802  4.35877811974283E-09
150   160.889205710213  -2.45893958744307E-09
151    84.4502033182748  -3.00133311935682E-09
152   -112.732145115534  -3.33652618216474E-10
153    4.227674507727702  4.07562229696174E-08
154  -10.8342582573892  -2.32586997681632E-09
155    4.9655805882639   -1.05611140468916E-07
156   -0.553850439419576  -1.98977945743534E-08
157   -8.204330403731144  -8.27105949058726E-08
158   114.83608474643  -2.48317911575925E-08
159  128.51819165101  -2.63457735094495E-09
160   13.7239100272455  -1.99531479697656E-09
161  -48.157060604333  -4.11525980456149E-09

Forecast

<table>
<thead>
<tr>
<th>Expected</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td>1</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>1</td>
</tr>
<tr>
<td>599</td>
<td>0</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
</tr>
<tr>
<td>600</td>
<td>0</td>
</tr>
<tr>
<td>601</td>
<td>0</td>
</tr>
<tr>
<td>602</td>
<td>0</td>
</tr>
<tr>
<td>603</td>
<td>0</td>
</tr>
<tr>
<td>604</td>
<td>0</td>
</tr>
<tr>
<td>605</td>
<td>0</td>
</tr>
<tr>
<td>606</td>
<td>0</td>
</tr>
<tr>
<td>607</td>
<td>0</td>
</tr>
<tr>
<td>608</td>
<td>0</td>
</tr>
<tr>
<td>609</td>
<td>0</td>
</tr>
<tr>
<td>610</td>
<td>0</td>
</tr>
<tr>
<td>611</td>
<td>0</td>
</tr>
<tr>
<td>612</td>
<td>0</td>
</tr>
<tr>
<td>613</td>
<td>0</td>
</tr>
<tr>
<td>614</td>
<td>0</td>
</tr>
<tr>
<td>615</td>
<td>0</td>
</tr>
<tr>
<td>616</td>
<td>0</td>
</tr>
<tr>
<td>617</td>
<td>0</td>
</tr>
<tr>
<td>618</td>
<td>0</td>
</tr>
<tr>
<td>619</td>
<td>0</td>
</tr>
<tr>
<td>620</td>
<td>0</td>
</tr>
<tr>
<td>621</td>
<td>0</td>
</tr>
<tr>
<td>622</td>
<td>0</td>
</tr>
<tr>
<td>623</td>
<td>0</td>
</tr>
<tr>
<td>624</td>
<td>0</td>
</tr>
<tr>
<td>625</td>
<td>0</td>
</tr>
<tr>
<td>626</td>
<td>0</td>
</tr>
<tr>
<td>627</td>
<td>0</td>
</tr>
<tr>
<td>628</td>
<td>0</td>
</tr>
<tr>
<td>629</td>
<td>0</td>
</tr>
<tr>
<td>630</td>
<td>0</td>
</tr>
<tr>
<td>631</td>
<td>0</td>
</tr>
<tr>
<td>632</td>
<td>0</td>
</tr>
<tr>
<td>633</td>
<td>0</td>
</tr>
<tr>
<td>634</td>
<td>0</td>
</tr>
<tr>
<td>635</td>
<td>0</td>
</tr>
<tr>
<td>636</td>
<td>0</td>
</tr>
<tr>
<td>637</td>
<td>0</td>
</tr>
<tr>
<td>638</td>
<td>0</td>
</tr>
<tr>
<td>639</td>
<td>0</td>
</tr>
<tr>
<td>640</td>
<td>0</td>
</tr>
<tr>
<td>641</td>
<td>0</td>
</tr>
<tr>
<td>642</td>
<td>0</td>
</tr>
<tr>
<td>643</td>
<td>0</td>
</tr>
<tr>
<td>644</td>
<td>0</td>
</tr>
<tr>
<td>645</td>
<td>0</td>
</tr>
<tr>
<td>646</td>
<td>0</td>
</tr>
<tr>
<td>647</td>
<td>0</td>
</tr>
<tr>
<td>648</td>
<td>0</td>
</tr>
<tr>
<td>649</td>
<td>0</td>
</tr>
<tr>
<td>650</td>
<td>0</td>
</tr>
<tr>
<td>651</td>
<td>0</td>
</tr>
<tr>
<td>652</td>
<td>0</td>
</tr>
<tr>
<td>653</td>
<td>0</td>
</tr>
<tr>
<td>654</td>
<td>0</td>
</tr>
</tbody>
</table>
MultiClassification Class

public class Imsl.DataMining.Neural.MultiClassification

Classifies patterns into three or more classes.

Extends neural network analysis to solving multi-classification problems. In these problems, the target output for the network is the probability that the pattern falls into each of several classes, where the number of classes is 3 or greater. These probabilities are then used to assign patterns to one of the target classes. Typical applications include determining the credit classification for a business (excellent, good, fair or poor), and determining which of three or more treatments a patient should receive based upon their physical, clinical and laboratory information. This class signals that network training will minimize the multi-classification cross-entropy error, and that network outputs are the probabilities that the pattern belongs to each of the target classes. These probabilities are scaled to sum to 1.0 using softmax activation.
Properties

**Error**

```csharp
virtual public Imsl.DataMining.Neural.QuasiNewtonTrainer.IError Error {get; }
```

**Description**
The error function for use by `QuasiNewtonTrainer` for training a classification network.

**Property Value**
An implementation of the multi-classification cross-entropy error function.

**Remarks**
This error function combines the softmax activation function and the cross-entropy error function.

**Network**

```csharp
virtual public Imsl.DataMining.Neural.Network Network {get; }
```

**Description**
Returns the network being used for classification.

Constructor

**MultiClassification**

```csharp
public MultiClassification(Imsl.DataMining.Neural.Network network)
```

**Description**
Creates a classifier.

**Parameter**
- `network` – Is the neural network used for classification. Its OutputPerceptrons should use linear activation functions, `Activation.Linear`. The number of output Perceptrons should equal the number of classes.

Methods

**ComputeStatistics**

```csharp
virtual public double[] ComputeStatistics(double[,] xData, int[] yData)
```

**Description**
Computes classification statistics for the supplied network patterns and their associated classifications.
Parameters

xData – A double matrix specifying the input training patterns. The number of columns in xData must equal the number of Nodes in the InputLayer.

yData – An int[] containing the output classification patterns. The values in yData must be in the range of one to the number of OutputPerceptrons in the network.

Returns

A double[] containing the cross-entropy error and the classification error rate.

Remarks

Method ComputeStatistics returns a two element array where the first element returned is the cross-entropy error; the second is the classification error rate. The classification error rate is calculated by comparing the estimated classification probabilities to the target classifications. If the estimated probability for the target class is not the largest among the target classes, then the pattern is tallied as a classification error.

PredictedClass

virtual public int PredictedClass(double[] x)

Description

Calculates the classification probabilities for the input pattern x, and returns the class with the highest probability.

Parameter

x – The double array containing the network input patterns to classify. The length of x should equal the number of inputs in the network.

Returns

The classification predicted by the trained network for x. This will be one of the integers 1,2,...,nClasses, where nClasses is equal to nOuptuts. nOuptuts is the number of outputs in the network representing the number of classes.

Remarks

This method classifies patterns into one of the target classes based upon the patterns values.

Probabilities

virtual public double[] Probabilities(double[] x)

Description

Returns classification probabilities for the input pattern x.

Parameter

x – A double array containing the input patterns to classify. The length of x must be equal to the number of input nodes.

Returns

A double containing the scaled probabilities.
Remarks
The number of probabilities is equal to the number of target classes, which is the number of outputs in the FeedForwardNetwork. Each are calculated using the softmax activation for each of the OutputPerceptrons. The softmax function transforms the outputs potential $z$ to the probability $y$ by

$$y_i = \text{softmax}_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

Train
virtual public void Train(Imsl.DataMining.Neural.ITrainer trainer, double[,] xData, int[] yData)

Description
Trains the classification neural network using supplied training patterns.

Parameters
- trainer – A Trainer object, which is used to train the network. The error function in any QuasiNewton trainer included in trainer should be set to the error function from this class using the Error (p. 1728) method.
- xData – A double matrix containing the input training patterns. The number of columns in xData must equal the number of nodes in the input layer. Each row of xData contains a training pattern.
- yData – An int array containing the output classification patterns. These values must be in the range of one to the number of output perceptrons in the network.

Example 1: MultiClassification
This example trains a 3-layer network using Fisher’s Iris data with four continuous input attributes and three output classifications. This is perhaps the best known database to be found in the pattern recognition literature. Fisher’s paper is a classic in the field. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

The structure of the network consists of four input nodes and three layers, with four perceptrons in the first hidden layer, three perceptrons in the second hidden layer and three in the output layer.

The four input attributes represent
1. Sepal length
2. Sepal width
3. Petal length
4. Petal width

The output attribute represents the class of the iris plant and are encoded using binary encoding.

1. Iris Setosa
2. Iris Versicolour

3. Iris Virginica

There are a total of 46 weights in this network, including the bias weights. All hidden layers use the logistic activation function. Since the target output is multi-classification the softmax activation function is used in the output layer and the MultiClassification error function class is used by the trainer. The error class MultiClassification combines the cross-entropy error calculations and the softmax function.

```csharp
using System;
using Imsl.DataMining.Neural;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;

//*****************************************************************************
// Three Layer Feed-Forward Network with 4 inputs, all
// continuous, and 3 classification categories.
//
// new classification training_ex5.c
//
// This is perhaps the best known database to be found in the pattern
// recognition literature. Fisher's paper is a classic in the field.
// The data set contains 3 classes of 50 instances each,
// where each class refers to a type of iris plant. One class is
// linearly separable from the other 2; the latter are NOT linearly
// separable from each other.
//
// Predicted attribute: class of iris plant.
// 1=Iris Setosa, 2=Iris Versicolour, and 3=Iris Virginica
//
// Input Attributes (4 Continuous Attributes)
// X1: Sepal length, X2: Sepal width, X3: Petal length, and X4: Petal width
//*****************************************************************************

[Serializable]
public class MultiClassificationEx1
{
    private static int nObs = 150; // number of training patterns
    private static int nInputs = 4; // 9 nominal coded as 0=x, 1=o, 2=blank
    private static int nOutputs = 3; // one continuous output (nClasses=2)

    // irisData[][]: The raw data matrix. This is a 2-D matrix with 150 rows and
    // 5 columns. The first 4 columns are the continuous input
    // attributes and the 5th column is the classification category
    // (1-3). These data contain no categorical input attributes.

    private static double[][] irisData = new double[][]{
        new double[] {5.1, 3.5, 1.4, 0.2, 1},
        new double[] {4.9, 3.0, 1.4, 0.2, 1},
        new double[] {4.7, 3.2, 1.3, 0.2, 1},
        new double[] {4.6, 3.1, 1.5, 0.2, 1},
        new double[] {5.0, 3.6, 1.4, 0.2, 1},
        new double[] {5.4, 3.9, 1.7, 0.4, 1},
        new double[] {4.6, 3.4, 1.4, 0.3, 1},
        new double[] {5.0, 3.4, 1.5, 0.2, 1},
        new double[] {4.4, 2.9, 1.4, 0.2, 1},
        new double[] {4.9, 3.1, 1.5, 0.1, 1},
```
new double[]{5.4, 3.7, 1.5, 0.2, 1}, new double[]{4.8, 3.4, 1.6, 0.2, 1},
new double[]{5.8, 4.0, 1.2, 0.2, 1}, new double[]{5.7, 4.4, 1.5, 0.4, 1},
new double[]{5.4, 3.9, 1.3, 0.4, 1}, new double[]{5.1, 3.5, 1.4, 0.3, 1},
new double[]{5.4, 3.4, 1.7, 0.2, 1}, new double[]{5.1, 3.8, 1.5, 0.3, 1},
new double[]{4.6, 3.6, 1.0, 0.2, 1}, new double[]{5.1, 3.7, 1.5, 0.4, 1},
new double[]{4.8, 3.4, 1.9, 0.2, 1}, new double[]{5.0, 3.0, 1.6, 0.2, 1},
new double[]{5.0, 3.4, 1.6, 0.4, 1}, new double[]{5.2, 3.5, 1.5, 0.2, 1},
new double[]{5.2, 4.1, 1.5, 0.1, 1}, new double[]{5.5, 4.2, 1.4, 0.2, 1},
new double[]{4.9, 3.1, 1.5, 0.1, 1}, new double[]{5.0, 3.2, 1.2, 0.2, 1},
new double[]{5.5, 3.5, 1.3, 0.2, 1}, new double[]{4.9, 3.1, 1.5, 0.1, 1},
new double[]{4.4, 3.0, 1.3, 0.2, 1}, new double[]{5.1, 3.4, 1.5, 0.2, 1},
new double[]{5.0, 3.5, 1.3, 0.3, 1}, new double[]{4.5, 2.3, 1.3, 0.3, 1},
new double[]{4.4, 3.2, 1.3, 0.2, 1}, new double[]{5.0, 3.5, 1.6, 0.6, 1},
new double[]{5.1, 3.8, 1.9, 0.4, 1}, new double[]{4.8, 3.0, 1.4, 0.3, 1},
new double[]{5.1, 3.8, 1.6, 0.2, 1}, new double[]{4.6, 3.2, 1.4, 0.2, 1},
new double[]{5.3, 3.7, 1.5, 0.2, 1}, new double[]{5.0, 3.3, 1.4, 0.2, 1},
new double[]{7.0, 3.2, 4.7, 1.4, 2}, new double[]{6.4, 3.2, 4.5, 1.5, 2},
new double[]{6.9, 3.1, 4.9, 1.5, 2}, new double[]{5.5, 2.3, 4.0, 1.3, 2},
new double[]{6.5, 2.8, 4.6, 1.5, 2}, new double[]{5.7, 2.8, 4.5, 1.3, 2},
new double[]{6.3, 3.3, 4.7, 1.6, 2}, new double[]{4.9, 2.4, 3.3, 1.0, 2},
new double[]{6.6, 2.9, 4.6, 1.3, 2}, new double[]{5.2, 2.7, 3.9, 1.4, 2},
new double[]{5.0, 2.0, 3.5, 1.0, 2}, new double[]{5.9, 3.0, 4.2, 1.5, 2},
new double[]{6.0, 2.2, 4.0, 1.0, 2}, new double[]{6.1, 2.9, 4.7, 1.4, 2},
new double[]{5.6, 2.9, 3.6, 1.3, 2}, new double[]{6.7, 3.1, 4.4, 1.4, 2},
new double[]{5.6, 3.0, 4.5, 1.5, 2}, new double[]{5.8, 2.7, 4.1, 1.0, 2},
new double[]{6.2, 2.2, 4.5, 1.5, 2}, new double[]{5.6, 2.5, 3.9, 1.1, 2},
new double[]{5.9, 3.2, 4.8, 1.8, 2}, new double[]{6.1, 2.8, 4.0, 1.3, 2},
new double[]{6.3, 2.5, 4.9, 1.5, 2}, new double[]{6.1, 2.8, 4.7, 1.2, 2},
new double[]{6.4, 2.9, 4.3, 1.3, 2}, new double[]{6.6, 3.0, 4.4, 1.4, 2},
new double[]{6.8, 2.8, 4.8, 1.4, 2}, new double[]{6.7, 3.0, 5.0, 1.7, 2},
new double[]{6.0, 2.9, 4.5, 1.5, 2}, new double[]{5.7, 2.6, 3.5, 1.0, 2},
new double[]{5.5, 2.4, 3.8, 1.1, 2}, new double[]{5.5, 2.4, 3.7, 1.0, 2},
new double[]{5.8, 2.7, 3.9, 1.2, 2}, new double[]{6.0, 2.7, 5.1, 1.6, 2},
new double[]{5.4, 3.0, 4.5, 1.5, 2}, new double[]{6.0, 3.4, 4.5, 1.6, 2},
new double[]{6.7, 3.1, 4.7, 1.5, 2}, new double[]{6.3, 2.3, 4.4, 1.3, 2},
new double[]{5.6, 3.0, 4.1, 1.3, 2}, new double[]{5.5, 2.5, 4.0, 1.3, 2},
new double[]{5.5, 2.6, 4.4, 1.2, 2}, new double[]{6.1, 3.0, 4.6, 1.4, 2},
new double[]{5.8, 2.6, 4.0, 1.2, 2}, new double[]{5.0, 2.3, 3.3, 1.0, 2},
new double[]{5.6, 2.7, 4.2, 1.3, 2}, new double[]{5.7, 3.0, 4.2, 1.2, 2},
new double[]{5.7, 2.9, 4.2, 1.3, 2}, new double[]{6.2, 2.9, 4.3, 1.3, 2},
new double[]{5.1, 2.5, 3.0, 1.1, 2}, new double[]{5.7, 2.8, 4.1, 1.3, 2},
new double[]{6.3, 3.3, 6.0, 2.5, 3}, new double[]{5.8, 2.7, 5.1, 1.9, 3},
new double[]{7.1, 3.0, 5.9, 2.1, 3}, new double[]{6.3, 2.9, 5.6, 1.8, 3},
new double[]{6.5, 3.0, 5.8, 2.2, 3}, new double[]{7.6, 3.0, 6.6, 2.1, 3},
new double[]{4.9, 2.5, 4.5, 1.7, 3}, new double[]{7.3, 2.9, 6.3, 1.8, 3},
new double[]{6.7, 2.5, 5.8, 1.8, 3}, new double[]{7.2, 3.6, 6.1, 2.5, 3},
new double[]{6.5, 3.2, 5.1, 2.0, 3}, new double[]{6.4, 2.7, 5.3, 1.9, 3},
new double[]{6.8, 3.0, 5.5, 2.1, 3}, new double[]{5.7, 2.5, 5.0, 2.0, 3},
new double[]{5.8, 2.8, 5.1, 2.4, 3}, new double[]{6.4, 3.2, 5.3, 2.3, 3},
new double[]{6.5, 3.0, 5.5, 1.8, 3}, new double[]{7.7, 3.8, 6.7, 2.2, 3},
new double[]{7.7, 2.6, 6.9, 2.3, 3}, new double[]{6.0, 2.2, 5.0, 1.5, 3},
new double[]{6.9, 3.2, 5.7, 2.3, 3}, new double[]{5.6, 2.8, 4.9, 2.0, 3}.
public static void Main(System.String[] args)
{
    double[,] xData = new double[nObs, nInputs];
    int[] yData = new int[nObs];

    for (int i = 0; i < nObs; i++)
    {
        for (int j = 0; j < nInputs; j++)
        {
            xData[i, j] = irisData[i][j];
        }
        yData[i] = (int) irisData[i][4];
    }

    // Create network
    FeedForwardNetwork network = new FeedForwardNetwork();
    network.InputLayer.CreateInputs(nInputs);
    network.CreateHiddenLayer().CreatePerceptrons(4,
        Imsl.DataMining.Neural.Activation.Logistic, 0.0);
    network.CreateHiddenLayer().CreatePerceptrons(3,
        Imsl.DataMining.Neural.Activation.Logistic, 0.0);
    network.OutputLayer.CreatePerceptrons(nOutputs,
        Imsl.DataMining.Neural.Activation.Softmax, 0.0);
    network.LinkAll();

    MultiClassification classification = new MultiClassification(network);

    // Create trainer
    QuasiNewtonTrainer trainer = new QuasiNewtonTrainer();
    trainer.SetError(classification.Error);
    trainer.MaximumTrainingIterations = 1000;

    // Train Network
    long t0 = (System.DateTime.Now.Ticks - 621355968000000000) / 10000;
    classification.Train(trainer, xData, yData);

    // Display Network Errors
    double[] stats = classification.ComputeStatistics(xData, yData);
    System.Console.Out.WriteLine( neural nets multiclassification • 1733

  new double[]{7.7, 2.8, 6.7, 2.0, 3}, new double[]{6.3, 2.7, 4.9, 1.8, 3},
  new double[]{6.7, 3.3, 5.7, 2.1, 3}, new double[]{7.2, 3.2, 6.0, 1.8, 3},
  new double[]{6.2, 2.8, 4.8, 1.8, 3}, new double[]{6.1, 3.0, 4.9, 1.8, 3},
  new double[]{6.4, 2.8, 5.6, 2.1, 3}, new double[]{7.2, 3.0, 5.8, 1.6, 3},
  new double[]{7.4, 2.8, 6.1, 1.9, 3}, new double[]{7.9, 3.8, 6.4, 2.0, 3},
  new double[]{6.4, 2.8, 5.6, 2.2, 3}, new double[]{6.3, 2.8, 5.1, 1.5, 3},
  new double[]{6.1, 2.6, 5.6, 1.4, 3}, new double[]{7.7, 3.0, 6.1, 2.3, 3},
  new double[]{6.3, 3.4, 5.6, 2.4, 3}, new double[]{6.4, 3.1, 5.5, 1.8, 3},
  new double[]{6.0, 3.0, 4.8, 1.8, 3}, new double[]{6.9, 3.1, 5.4, 2.1, 3},
  new double[]{6.7, 3.1, 5.6, 2.4, 3}, new double[]{6.9, 3.1, 5.1, 2.3, 3},
  new double[]{5.8, 2.7, 5.1, 1.9, 3}, new double[]{6.8, 3.2, 5.9, 2.3, 3},
  new double[]{6.7, 3.3, 5.7, 2.5, 3}, new double[]{6.7, 3.0, 5.2, 2.3, 3},
  new double[]{6.3, 2.5, 5.0, 1.9, 3}, new double[]{6.5, 3.0, 5.2, 2.0, 3},
  new double[]{6.2, 3.4, 5.4, 2.3, 3}, new double[]{5.9, 3.0, 5.1, 1.8, 3});
System.Console.Out.WriteLine("--> Cross-entropy error: " + (float) stats[0]);
System.Console.Out.WriteLine("--> Classification error rate: " + (float) stats[1]);
System.Console.Out.WriteLine("***********************************************");
System.Console.Out.WriteLine("\n");

double[] weight = network.Weights;
double[] gradient = trainer.ErrorGradient;
double[][] wg = new double[weight.Length][];
for (int i2 = 0; i2 < weight.Length; i2++)
{
    wg[i2] = new double[2];
}
for (int i = 0; i < weight.Length; i++)
{
    wg[i][0] = weight[i];
    wg[i][1] = gradient[i];
}  
PrintMatrixFormat pmf = new PrintMatrixFormat();
pmf.SetColumnLabels(new System.String[] {"Weights", "Gradients"});
new PrintMatrix().Print(pmf, wg);

double[][] report = new double[nObs][];
for (int i3 = 0; i3 < nObs; i3++)
{
    report[i3] = new double[nInputs + 2];
}
for (int i = 0; i < nObs; i++)
{
    for (int j = 0; j < nInputs; j++)
    {
        report[i][j] = xData[i,j];
    }
    report[i][nInputs] = irisData[i][4];
    double[] xTmp = new double[xData.GetLength(1)];
    for (int j=0; j<xData.GetLength(1); j++)
        xTmp[j] = xData[i,j];
    report[i][nInputs + 1] = classification.PredictedClass(xTmp);
}  

pmf = new PrintMatrixFormat();
new PrintMatrix("Forecast").Print(pmf, report);

// ********************************************************************************
// DISPLAY CLASSIFICATION STATISTICS
// ********************************************************************************

double[] statsClass = classification.ComputeStatistics(xData, yData);
System.Console.Out.WriteLine("\n");
Output

***********************************************
--> Cross-entropy error: 5.922989
--> Classification error rate: 0.0133333
***********************************************

<table>
<thead>
<tr>
<th></th>
<th>Weights</th>
<th>Gradients</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-362.135397018074</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-560.715494616693</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>527.72406779352</td>
<td>2.08315681881952E-198</td>
</tr>
<tr>
<td>3</td>
<td>-0.195010974908058</td>
<td>0.0264624672168392</td>
</tr>
<tr>
<td>4</td>
<td>-685.159411754574</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>-803.111185471252</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>-195.048774132049</td>
<td>9.50211882268555E-199</td>
</tr>
<tr>
<td>7</td>
<td>-0.649536948895873</td>
<td>0.0122051719495706</td>
</tr>
<tr>
<td>8</td>
<td>1271.3576881191</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>495.146969161579</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>-599.774609230401</td>
<td>1.2791313799769E-198</td>
</tr>
<tr>
<td>11</td>
<td>0.807201787624182</td>
<td>0.0211017819573165</td>
</tr>
<tr>
<td>12</td>
<td>1143.16439557087</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>950.665441902952</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>-501.328120816785</td>
<td>3.654661085648292E-199</td>
</tr>
<tr>
<td>15</td>
<td>1.82418587547183</td>
<td>0.0071872906296363</td>
</tr>
<tr>
<td>16</td>
<td>188.142460610324</td>
<td>0.606686791122197</td>
</tr>
<tr>
<td>17</td>
<td>99.9378595651514</td>
<td>4.4316403183572E-05</td>
</tr>
<tr>
<td>18</td>
<td>48.0430167242241</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>103.388473027949</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>-28.058983107429</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>-122.247432161144</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>674.932369179828</td>
<td>5.4146828969623E-202</td>
</tr>
<tr>
<td>23</td>
<td>295.103891118948</td>
<td>3.9419889629911E-206</td>
</tr>
<tr>
<td>24</td>
<td>-885.621857738157</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>31.424095865498</td>
<td>0.000136908693839501</td>
</tr>
<tr>
<td>26</td>
<td>-11.7156988030889</td>
<td>9.32263272488116E-09</td>
</tr>
<tr>
<td>27</td>
<td>250.928447008145</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>-9490.85347121632</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>1392.60893930402</td>
<td>-0.000263660213410609</td>
</tr>
<tr>
<td>30</td>
<td>8099.24453182847</td>
<td>0.000263660213411595</td>
</tr>
</tbody>
</table>

Neural Nets MultiClassification • 1735
### Forecast

<table>
<thead>
<tr>
<th>Sepal Length</th>
<th>Sepal Width</th>
<th>Petal Length</th>
<th>Petal Width</th>
<th>Expected</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.1</td>
<td>1.4</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>1.4</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4.7</td>
<td>1.3</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4.6</td>
<td>1.5</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1.4</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5.4</td>
<td>1.7</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>4.6</td>
<td>1.4</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>1.5</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>4.4</td>
<td>1.4</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4.9</td>
<td>1.5</td>
<td>0.1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>5.4</td>
<td>1.5</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>4.8</td>
<td>1.6</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>4.8</td>
<td>1.4</td>
<td>0.1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>4.3</td>
<td>1.1</td>
<td>0.1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>5.8</td>
<td>1.2</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>5.7</td>
<td>1.5</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>5.4</td>
<td>1.3</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>5.1</td>
<td>1.4</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>5.7</td>
<td>1.7</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>5.1</td>
<td>1.5</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>5.4</td>
<td>1.7</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>5.1</td>
<td>1.5</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>4.6</td>
<td>1</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>5.1</td>
<td>1.7</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>4.8</td>
<td>1.9</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td>1.6</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>5</td>
<td>1.6</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>5.2</td>
<td>1.5</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>5.2</td>
<td>1.4</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>4.7</td>
<td>1.6</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>4.8</td>
<td>1.6</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>5.4</td>
<td>1.5</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>5.2</td>
<td>1.5</td>
<td>0.1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>5.5</td>
<td>1.4</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>4.9</td>
<td>1.5</td>
<td>0.1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>35</td>
<td>5</td>
<td>1.2</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td>5.5</td>
<td>1.3</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>37</td>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>4.4</td>
<td>3</td>
<td>1.3</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>39</td>
<td>5.1</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>3.5</td>
<td>1.3</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>4.5</td>
<td>2.3</td>
<td>1.3</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>4.4</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>5</td>
<td>3.5</td>
<td>1.6</td>
<td>0.6</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>5.1</td>
<td>3.8</td>
<td>1.9</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>45</td>
<td>4.8</td>
<td>3</td>
<td>1.4</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>46</td>
<td>5.1</td>
<td>3.8</td>
<td>1.6</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>47</td>
<td>4.6</td>
<td>3.2</td>
<td>1.4</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>48</td>
<td>5.3</td>
<td>3.7</td>
<td>1.5</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>49</td>
<td>5</td>
<td>3.3</td>
<td>1.4</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>7</td>
<td>3.2</td>
<td>4.7</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>51</td>
<td>6.4</td>
<td>3.2</td>
<td>4.5</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>52</td>
<td>6.9</td>
<td>3.1</td>
<td>4.9</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>53</td>
<td>5.5</td>
<td>2.3</td>
<td>4</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>54</td>
<td>6.5</td>
<td>2.8</td>
<td>4.6</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>55</td>
<td>5.7</td>
<td>2.8</td>
<td>4.5</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>56</td>
<td>6.3</td>
<td>3.3</td>
<td>4.7</td>
<td>1.6</td>
<td>2</td>
</tr>
<tr>
<td>57</td>
<td>4.9</td>
<td>2.4</td>
<td>3.3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>58</td>
<td>6.6</td>
<td>2.9</td>
<td>4.6</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>59</td>
<td>5.2</td>
<td>2.7</td>
<td>3.9</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>60</td>
<td>5</td>
<td>2</td>
<td>3.5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>61</td>
<td>5.9</td>
<td>3</td>
<td>4.2</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>62</td>
<td>6</td>
<td>2.2</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>63</td>
<td>6.1</td>
<td>2.9</td>
<td>4.7</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>64</td>
<td>5.6</td>
<td>2.9</td>
<td>3.6</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>65</td>
<td>6.7</td>
<td>3.1</td>
<td>4.4</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>66</td>
<td>5.6</td>
<td>3</td>
<td>4.5</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>67</td>
<td>5.8</td>
<td>2.7</td>
<td>4.1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>68</td>
<td>6.2</td>
<td>2.2</td>
<td>4.5</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>69</td>
<td>5.6</td>
<td>2.5</td>
<td>3.9</td>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>70</td>
<td>5.9</td>
<td>3.2</td>
<td>4.8</td>
<td>1.8</td>
<td>2</td>
</tr>
<tr>
<td>71</td>
<td>6.1</td>
<td>2.8</td>
<td>4</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>72</td>
<td>6.3</td>
<td>2.5</td>
<td>4.9</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>73</td>
<td>6.1</td>
<td>2.8</td>
<td>4.7</td>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>74</td>
<td>6.4</td>
<td>2.9</td>
<td>4.3</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>75</td>
<td>6.6</td>
<td>3</td>
<td>4.4</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>76</td>
<td>6.8</td>
<td>2.8</td>
<td>4.8</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>77</td>
<td>6.7</td>
<td>3</td>
<td>5</td>
<td>1.7</td>
<td>2</td>
</tr>
<tr>
<td>78</td>
<td>6</td>
<td>2.9</td>
<td>4.5</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>79</td>
<td>5.7</td>
<td>2.6</td>
<td>3.5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>80</td>
<td>5.5</td>
<td>2.4</td>
<td>3.8</td>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>81</td>
<td>5.5</td>
<td>2.4</td>
<td>3.7</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>82</td>
<td>5.8</td>
<td>2.7</td>
<td>3.9</td>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>83</td>
<td>6</td>
<td>2.7</td>
<td>5.1</td>
<td>1.6</td>
<td>2</td>
</tr>
<tr>
<td>84</td>
<td>5.4</td>
<td>3</td>
<td>4.5</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>85</td>
<td>6</td>
<td>3.4</td>
<td>4.5</td>
<td>1.6</td>
<td>2</td>
</tr>
<tr>
<td>86</td>
<td>6.7</td>
<td>3.1</td>
<td>4.7</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>87</td>
<td>6.3</td>
<td>2.3</td>
<td>4.4</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>88</td>
<td>5.6</td>
<td>3</td>
<td>4.1</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>89</td>
<td>5.5</td>
<td>2.5</td>
<td>4</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>90</td>
<td>5.5</td>
<td>2.6</td>
<td>4.4</td>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>91</td>
<td>6.1</td>
<td>3</td>
<td>4.6</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>92</td>
<td>5.8</td>
<td>2.6</td>
<td>4</td>
<td>1.2</td>
<td>2</td>
</tr>
</tbody>
</table>

Neural Nets  MultiClassification • 1737
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
<td>5</td>
<td>2.3</td>
<td>3.3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>94</td>
<td>5.6</td>
<td>2.7</td>
<td>4.2</td>
<td>1.3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>95</td>
<td>5.7</td>
<td>2.4</td>
<td>4.2</td>
<td>1.3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>96</td>
<td>5.7</td>
<td>2.9</td>
<td>4.3</td>
<td>1.3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>97</td>
<td>6.2</td>
<td>2.9</td>
<td>4.3</td>
<td>1.3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>98</td>
<td>5.1</td>
<td>2.5</td>
<td>3</td>
<td>1.1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>99</td>
<td>5.7</td>
<td>2.8</td>
<td>4.1</td>
<td>1.3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>6.3</td>
<td>3.3</td>
<td>6</td>
<td>2.5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>101</td>
<td>5.8</td>
<td>2.7</td>
<td>5.1</td>
<td>1.9</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>102</td>
<td>7.1</td>
<td>3</td>
<td>5.9</td>
<td>2.1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>103</td>
<td>6.3</td>
<td>2.9</td>
<td>5.6</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>104</td>
<td>6.5</td>
<td>3</td>
<td>5.8</td>
<td>2.2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>105</td>
<td>7.6</td>
<td>3</td>
<td>4.6</td>
<td>2.1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>106</td>
<td>4.9</td>
<td>2.5</td>
<td>4.5</td>
<td>1.7</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>107</td>
<td>7.3</td>
<td>2.9</td>
<td>6.3</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>108</td>
<td>6.7</td>
<td>2.5</td>
<td>5.8</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>109</td>
<td>7.2</td>
<td>3.6</td>
<td>6.1</td>
<td>2.5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>110</td>
<td>6.5</td>
<td>3.2</td>
<td>5.1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>111</td>
<td>6.4</td>
<td>2.7</td>
<td>5.3</td>
<td>1.9</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>112</td>
<td>6.8</td>
<td>3</td>
<td>5.5</td>
<td>2.1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>113</td>
<td>5.7</td>
<td>2.5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>114</td>
<td>5.8</td>
<td>2.8</td>
<td>5.1</td>
<td>2.4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>115</td>
<td>6.4</td>
<td>3.2</td>
<td>5.3</td>
<td>2.3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>116</td>
<td>6.5</td>
<td>3</td>
<td>5.5</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>117</td>
<td>7.7</td>
<td>3.8</td>
<td>6.7</td>
<td>2.2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>118</td>
<td>7.7</td>
<td>2.6</td>
<td>6.9</td>
<td>2.3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>119</td>
<td>6</td>
<td>2.2</td>
<td>5</td>
<td>1.5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>120</td>
<td>6.9</td>
<td>3.2</td>
<td>5.7</td>
<td>2.3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>121</td>
<td>5.6</td>
<td>2.8</td>
<td>4.9</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>122</td>
<td>7.7</td>
<td>2.8</td>
<td>6.7</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>123</td>
<td>6.3</td>
<td>2.7</td>
<td>4.9</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>124</td>
<td>6.7</td>
<td>3.3</td>
<td>5.7</td>
<td>2.1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>125</td>
<td>7.2</td>
<td>3.2</td>
<td>6</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>126</td>
<td>6.2</td>
<td>2.8</td>
<td>4.8</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>127</td>
<td>6.1</td>
<td>3</td>
<td>4.9</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>128</td>
<td>6.4</td>
<td>2.8</td>
<td>5.6</td>
<td>2.1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>129</td>
<td>7.2</td>
<td>3</td>
<td>5.8</td>
<td>1.6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>130</td>
<td>7.4</td>
<td>2.8</td>
<td>6.1</td>
<td>1.9</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>131</td>
<td>7.9</td>
<td>3.8</td>
<td>6.4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>132</td>
<td>6.4</td>
<td>2.8</td>
<td>5.6</td>
<td>2.2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>133</td>
<td>6.3</td>
<td>2.8</td>
<td>5.1</td>
<td>1.5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>134</td>
<td>6.1</td>
<td>2.6</td>
<td>5.6</td>
<td>1.4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>135</td>
<td>7.7</td>
<td>3</td>
<td>6.1</td>
<td>2.3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>136</td>
<td>6.3</td>
<td>3.4</td>
<td>5.6</td>
<td>2.4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>137</td>
<td>6.4</td>
<td>3.1</td>
<td>5.5</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>138</td>
<td>6</td>
<td>3</td>
<td>4.8</td>
<td>1.8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>139</td>
<td>6.9</td>
<td>3.1</td>
<td>5.4</td>
<td>2.1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>140</td>
<td>6.7</td>
<td>3.1</td>
<td>5.6</td>
<td>2.4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>141</td>
<td>6.9</td>
<td>3.1</td>
<td>5.1</td>
<td>2.3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>142</td>
<td>5.8</td>
<td>2.7</td>
<td>5.1</td>
<td>1.9</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>143</td>
<td>6.8</td>
<td>3.2</td>
<td>5.9</td>
<td>2.3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>144</td>
<td>6.7</td>
<td>3.3</td>
<td>5.7</td>
<td>2.5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>145</td>
<td>6.7</td>
<td>3</td>
<td>5.2</td>
<td>2.3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>146</td>
<td>6.3</td>
<td>2.5</td>
<td>5</td>
<td>1.9</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>147</td>
<td>6.5</td>
<td>3</td>
<td>5.2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>148</td>
<td>6.2</td>
<td>3.4</td>
<td>5.4</td>
<td>2.3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Example 2: MultiClassification

This example trains a 2-layer network using three binary inputs (X0, X1, X2) and one three-level classification (Y). Where

\[ Y = 0 \text{ if } X1 = 1 \]
\[ Y = 1 \text{ if } X2 = 1 \]
\[ Y = 2 \text{ if } X3 = 1 \]

```csharp
using System;
using Imsl.DataMining.Neural;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;
//****************************************************************************
// Two-Layer FFN with 3 binary inputs (X0, X1, X2) and one three-level 
// classification variable (Y)
// Y = 0 if X1 = 1
// Y = 1 if X2 = 1
// Y = 2 if X3 = 1
// (training_ex6)
//****************************************************************************

[Serializable]
public class MultiClassificationEx2
{
    private static int nObs = 6; // number of training patterns
    private static int nInputs = 3; // 3 inputs, all categorical
    private static int nOutputs = 3;
    private static double[,] xData = {{1, 0, 0}, {1, 0, 0}, {0, 1, 0}, {0, 1, 0},
                                      {0, 0, 1}, {0, 0, 1}};
    private static int[] yData = new int[] {1, 1, 2, 2, 3, 3};
    private static double[] weights = new double[]{1.29099448735805800000,
                                                   -0.64549722436790290000,
                                                   0.00000000000000000000, 1.11803398874989490000,
                                                   -0.00000000000000005851,
                                                   -0.57735026918962573000,
                                                   0.00000000000000000000, 0.00000000000000000000, 0.00000000000000000000, 0.00000000000000000000,};

```
public static void Main(System.String[] args)
{
    FeedForwardNetwork network = new FeedForwardNetwork();
    network.InputLayer.CreateInputs(nInputs);
    network.CreateHiddenLayer().CreatePerceptrons(3,
        Imsl.DataMining.Neural.Activation.Linear, 0.0);
    network.OutputLayer.CreatePerceptrons(nOutputs,
        Imsl.DataMining.Neural.Activation.Softmax, 0.0);
    network.LinkAll();
    network.Weights = weights;

    MultiClassification classification = new MultiClassification(network);
    QuasiNewtonTrainer trainer = new QuasiNewtonTrainer();
    trainer.SetError(classification.Error);
    trainer.MaximumTrainingIterations = 1000;
    trainer.GradientTolerance = 1.0e-20;
    trainer.StepTolerance = 1.0e-20;

    // Train Network
    classification.Train(trainer, xData, yData);

    // Display Network Errors
    double[] stats = classification.ComputeStatistics(xData, yData);
    System.Console.Out.WriteLine(
        "************************************************************************");
    System.Console.Out.WriteLine(
        "--> Cross-Entropy Error: " + (float) stats[0]);
    System.Console.Out.WriteLine(
        "--> Classification Error: " + (float) stats[1]);
    System.Console.Out.WriteLine(
        "************************************************************************");
    System.Console.Out.WriteLine();

    double[] weight = network.Weights;
    double[] gradient = trainer.ErrorGradient;
    double[][] wg = new double[weight.Length][];
    for (int i = 0; i < weight.Length; i++)
    {
        wg[i] = new double[2];
    }
    for (int i = 0; i < weight.Length; i++)
    {
        wg[i][0] = weight[i];
        wg[i][1] = gradient[i];
    }
    PrintMatrixFormat pmf = new PrintMatrixFormat();
    pmf.SetColumnLabels(new System.String[] {"Weights", "Gradients"});
    new PrintMatrix().Print(pmf, wg);

    double[][] report = new double[nObs][];
    for (int i2 = 0; i2 < nObs; i2++)
    {
        report[i2] = new double[nInputs + nOutputs + 2];
    }
}
for (int i = 0; i < nObs; i++)
{
    for (int j = 0; j < nInputs; j++)
    {
        report[i][j] = xData[i,j];
    }
    report[i][nInputs] = yData[i];
    double[] xTmp = new double[xData.GetLength(1)];
    for (int j=0; j<xData.GetLength(1); j++)
        xTmp[j] = xData[i,j];
    double[] p = classification.Probabilities(xTmp);
    for (int j = 0; j < nOutputs; j++)
    {
        report[i][nInputs + 1 + j] = p[j];
    }
    report[i][nInputs + nOutputs + 1] = classification.PredictedClass(xTmp);
}

pmf = new PrintMatrixFormat();
pmf.SetColumnLabels(new System.String[]{"X1", "X2", "X3", "Y", "P(C1)", "P(C2)", "P(C3)", "Predicted"});
new PrintMatrix("Forecast").Print(pmf, report);
System.Console.Out.WriteLine("Cross-Entropy Error Value = " +
    trainer.ErrorValue);

// ******************************************************************************
// DISPLAY CLASSIFICATION STATISTICS
// ******************************************************************************
double[] statsClass = classification.ComputeStatistics(xData, yData);
// Display Network Errors
    (float)statsClass[0]);
System.Console.Out.WriteLine("Classification Error: " +
    (float)statsClass[1]);

Output

*****************************************************
--> Cross-Entropy Error: 0
--> Classification Error: 0
*****************************************************

Weights  Gradients
0  3.2662441281062 -1.07718767511655E-21
1 -3.65785075719586 9.08961951958712E-21
2 -1.34287873934408 4.06084823246345E-20
3 -1.35223399551086 1.06265766864622E-14

Neural Nets MultiClassification ● 1741
4  2.685355292965218  -7.71219393879981E-15
5  -2.87468867482566  6.1442307941349E-14
6  0.763542972743849  3.50677840927428E-42
7  2.27632473750162  -4.55192603059466E-42
8  5.01937720316383  -2.3312956152099E-42
9  4.5457678825492  -3.19544164040846E-18
10  -3.270345073713  1.92921479004263E-15
11  -0.2754220883618  -2.00863231270801E-17
12  -5.048518354049  1.72804459210703E-17
13  4.58077622000282  -1.04328902233367E-14
14  2.32407561540204  1.08624405719788E-14
15  -2.89362789116003  -8.61521051367963E-18
16  -6.4893352039981  5.20134410597363E-15
17  10.3829630951641  -5.4155039842253E-15
18  0.809208307413763  1.06265756092745E-14
19  0.251242239686325  -7.7121846981029E-15
20  1.41064046373263  6.1442714026172E-14
21  0.278456324867293  5.8845128569608E-18
22  0.345102584752059  -3.55271367880049E-15
23  -0.623585890961325  3.69899671823322E-15

Cross-Entropy Error Value = 0
*****************************************************************************
--> Cross-Entropy Error: 0
--> Classification Error: 0
*****************************************************************************

## ScaleFilter Class

```csharp
public class Imsl.DataMining.Neural.ScaleFilter
```

Scales or unscales continuous data prior to its use in neural network training, testing, or forecasting.

Bounded scaling is used to ensure that the values in the scaled array fall between a lower and upper bound. The scale limits have the following interpretation:
<table>
<thead>
<tr>
<th>Argument</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>realMin</td>
<td>The lowest value expected in x.</td>
</tr>
<tr>
<td>realMax</td>
<td>The largest value expected in x.</td>
</tr>
<tr>
<td>targetMin</td>
<td>The lower bound for the values in the scaled data.</td>
</tr>
<tr>
<td>targetMax</td>
<td>The upper bound for the values in the scaled data.</td>
</tr>
</tbody>
</table>

The scale limits are set using the method `SetBounds` (p. 1747).

The specific scaling used is controlled by the argument `scalingMethod` used when constructing the filter object. If `scalingMethod` is `ScalingMethod.None`, then no scaling is performed on the data.

If the input parameter `scalingMethod` is `ScaleMethod.Bounded` then the bounded method of scaling and unscaling is applied to x. The scaling operation is conducted using the scale limits set in method `SetBounds`, using the following calculation:

\[ z = r(x - \text{realMin}) + \text{targetMin}, \]

where

\[ r = \frac{\text{targetMax} - \text{targetMin}}{\text{realMax} - \text{realMin}}. \]

If `scalingMethod` is one of `UnboundedZScoreMeanStdev`, `UnboundedZScoreMedianMAD`, `BoundedZScoreMeanStdev`, or `BoundedZScoreMedianMAD`, then the z-score method of scaling is used. These calculations are based upon the following scaling calculation:

\[ z = \frac{(x - a)}{b}, \]

where \( a \) is a measure of center for \( x \), and \( b \) is a measure of the spread of \( x \).

If `scalingMethod` is `UnboundedZScoreMeanStdev`, or `BoundedZScoreMeanStdev`, then \( a \) and \( b \) are the arithmetic average and sample standard deviation of the training data.

If `scalingMethod` is `UnboundedZScoreMedianMAD` or `BoundedZScoreMedianMAD`, then \( a \) and \( b \) are the median and \( \tilde{s} \), where \( \tilde{s} \) is a robust estimate of the population standard deviation:

\[ \tilde{s} = \frac{\text{MAD}}{0.6745} \]

where \( \text{MAD} \) is the Mean Absolute Deviation

\[ \text{MAD} = \text{median}\{|x - \text{median}\{x\}|\} \]

The Mean Absolute Deviation is a robust measure of spread calculated by finding the median of the absolute value of differences between each non-missing value for the \( i \)-th variable and the median of those values.
If the method Decode (p. 1745) is called then an unscaling operation is conducted by inverting using:

\[ x = \frac{(z - \text{targetMin})}{r} + \text{realMin}. \]

**Unbounded z-score Scaling**

If scalingMethod is UnboundedZScoreMeanStdev or UnboundedZScoreMedianMAD, then a scaling operation is conducted using the z-score calculation:

\[ z = \frac{(x - \text{center})}{\text{spread}}, \]

If scalingMethod is UnboundedZScoreMeanStdev then Center (p. 1744) is set equal to the arithmetic average \( \bar{x} \) of \( x \), and Spread (p. 1745) is set equal to the sample standard deviation of \( x \). If scalingMethod is UnboundedZScoreMedianMAD then Center is set equal to the median \( \tilde{m} \) of \( x \), and Spread is set equal to the Mean Absolute Difference (MAD).

The method Decode can be used to unfilter data using the inverse calculation for the above equation:

\[ x = \text{spread} \cdot z + \text{center}. \]

**Bounded z-score Scaling**

This method is essentially the same as the z-score calculation described above with additional scaling or unscaling using the scale limits set in method SetBounds. The scaling operation is conducted using the well known z-score calculation:

\[ z = \frac{r \cdot (x - \text{center})}{\text{spread}} - r \cdot \text{realMin} + \text{targetMin}. \]

If scalingMethod is UnboundedZScoreMeanStdev then Center is set equal to the arithmetic average \( \bar{x} \) of \( x \), and Spread is set equal to the sample standard deviation of \( x \). If scalingMethod is UnboundedZScoreMedianMAD then Center is set equal to the median \( \tilde{m} \) of \( x \), and Spread is set equal to the Mean Absolute Difference (MAD).

The method Decode can be used to unfilter data using the inverse calculation for the above equation:

\[ x = \frac{\text{spread} \cdot (z - \text{targetMin})}{r} + \text{spread} \cdot \text{realMin} + \text{center} \]

**Properties**

**Center**

virtual public double Center {get; set;}
Description
The measure of center to be used during z-score scaling.

Property Value
A double containing the measure of center to be used during scaling.

Remarks
If this property is not set then the measure of center is computed from the data.

Spread
virtual public double Spread {get; set; }

Description
The measure of spread to be used during z-score scaling.

Property Value
A double containing the measure of spread to be used during z-score scaling.

Remarks
If this property is not set then the measure of spread is computed from the data.

Constructor


Description
Constructor for ScaleFilter.

Parameter
scalingMethod – An int specifying the scaling method to be applied.

Remarks

Methods

Decode
virtual public double Decode(double z)
**Description**
Unscales a value.

**Parameter**
- `z` – A double containing the value to be unscaled.

**Returns**
A double containing the filtered data.

**Decode**
```csharp
virtual public double[] Decode(double[] z)
```

**Description**
Unscales an array of values.

**Parameter**
- `z` – A double array of values to be unscaled.

**Returns**
A double array containing the filtered data.

**Decode**
```csharp
virtual public void Decode(int columnIndex, double[,] z)
```

**Description**
Unscales a single column of a two dimensional array of values.

**Parameters**
- `columnIndex` – An int specifying the index of the column of `z` to unscale.
- `z` – A double matrix containing the values to be unscaled.

**Remarks**
Indexing is zero-based. Its `columnIndex`-th column is modified in place.

**Encode**
```csharp
virtual public double Encode(double x)
```

**Description**
Scales a value.

**Parameter**
- `x` – A double containing the value to be scaled.

**Returns**
A double containing the scaled value.

**Encode**
```csharp
virtual public double[] Encode(double[] x)
```
Description
Scales an array of values.

Parameter
\(x\) – A double array containing the data to be scaled.

Returns
A double array containing the scaled data.

Encode
virtual public void Encode(int columnIndex, double[,] x)

Description
Scales a single column of a two dimensional array of values.

Parameters
\(\text{columnIndex}\) – An int specifying the index of the column of \(x\) to scale.
\(x\) – A double matrix containing the value to be scaled.

Remarks
Indexing is zero-based. Its columnIndex-th column is modified in place.

GetBounds
virtual public double[] GetBounds()

Description
Retrieves bounds used during bounded scaling.

Returns
A double array of length 4 containing the bounds.

Remarks

<table>
<thead>
<tr>
<th>(i)</th>
<th>(\text{result}[b])</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(\text{realMin}). Lowest expected value in the data to be filtered.</td>
</tr>
<tr>
<td>1</td>
<td>(\text{realMax}). Largest expected value in the data to be filtered.</td>
</tr>
<tr>
<td>2</td>
<td>(\text{targetMin}). Lowest allowed value in the filtered data.</td>
</tr>
<tr>
<td>3</td>
<td>(\text{targetMax}). Largest allowed value in the filtered data.</td>
</tr>
</tbody>
</table>

SetBounds
virtual public void SetBounds(double realMin, double realMax, double targetMin, double targetMax)

Description
Sets bounds to be used during bounded scaling and unscaling.
Parameters

realMin – A double containing the lowest expected value in the data to be filtered.
realMax – A double containing the largest expected value in the data to be filtered.
targetMin – A double containing the lowest allowed value in the filtered data.
targetMax – A double containing the largest allowed value in the filtered data.

Remarks

This method is normally called prior to calls to Encode or Decode. Otherwise the default bounds are realMin = 0, realMax = 1, targetMin = 0, and targetMax = 1. These bounds are ignored for unbounded scaling.

Example: ScaleFilter

In this example three sets of data, \( X_0 \), \( X_1 \), and \( X_2 \) are scaled using the methods described in the following table:

Variables and Scaling Methods

<table>
<thead>
<tr>
<th>Variable</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_0 )</td>
<td>0</td>
<td>No Scaling</td>
</tr>
<tr>
<td>( X_1 )</td>
<td>4</td>
<td>Bounded Z-score scaling using the mean and standard deviation of ( X_1 )</td>
</tr>
<tr>
<td>( X_2 )</td>
<td>5</td>
<td>Bounded Z-score scaling using the median and MAD of ( X_2 )</td>
</tr>
</tbody>
</table>

The bounds, measures of center and spread for \( X_1 \) and \( X_2 \) are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Real Limits</th>
<th>Target Limits</th>
<th>Measure of Center</th>
<th>Measure of Spread</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_1 )</td>
<td>(-6, +6)</td>
<td>(-3, +3)</td>
<td>3.4 (Mean)</td>
<td>1.7421 (Std. Dev.)</td>
</tr>
<tr>
<td>( X_2 )</td>
<td>(-3, +3)</td>
<td>(-3, +3)</td>
<td>2.4 (Median)</td>
<td>1.3343 (MAD/0.6745)</td>
</tr>
</tbody>
</table>

The real and target limits are used for bounded scaling. The measures of center and spread are used to calculate z-scores. Using these values for \( x_1[0]=3.5 \) yields the following calculations:

For \( x_1[0] \), the scale factor is calculated using the real and target limits in the above table:

\[ r = (3-(-3))/(6-(-6)) = 0.5 \]

The z-score for \( x_1[0] \) is calculated using the measures of center and spread:

\[ z_1[0] = (3.5 - 3.4)/1.7421 = 0.057402 \]

Since method=4 is used for \( x_1 \), this z-score is bounded (scaled) using the real and target limits:

\[ z_1(bounded) = r(z_1[0]) - r(realMin) + (targetMin) \]

\[ = 0.5(0.057402) - 0.5(-6) + (-3) = 0.029 \]

The calculations for \( x_2[0] \) are nearly identical, except that since method=5 for \( x_2 \), the median and MAD replace the mean and standard deviation used to calculate \( z_1(bounded) \):

1748 • ScaleFilter  IMSL C# Numerical Library
\[ r = \frac{(3-(-3))(3-(-3))}{(3-(-3))} = 1, \]

\[ z_2[0] = \frac{(3.1 - 2.4)}{1.3343} = 0.525, \text{ and} \]

\[ z_2(\text{bounded}) = r(z_2[0]) - r(\text{realMin}) + (\text{targetMin}) \]

\[ = 1(0.525) - 1(-3) + (-3) = 0.525 \]

Using System;
Using Imsl.Stat;
Using Imsl.Math;
Using Imsl.DataMining.Neural;

```csharp
public class ScaleFilterEx1
{
    public static void Main(System.String[] args)
    {
        ScaleFilter[] scaleFilter = new ScaleFilter[3];
        scaleFilter[0] = new ScaleFilter(ScaleFilter.ScalingMethod.None);
        scaleFilter[1] = new ScaleFilter(ScaleFilter.ScalingMethod.BoundedZScoreMeanStdev);
        scaleFilter[1].SetBounds(-6.0, 6.0, -3.0, 3.0);
        scaleFilter[2] = new ScaleFilter(ScaleFilter.ScalingMethod.BoundedZScoreMedianMAD);
        scaleFilter[2].SetBounds(-3.0, 3.0, -3.0, 3.0);
        double[] y0, y1, y2;
        double[] x0 = new double[]{1.2, 0.0, -1.4, 1.5, 3.2};
        double[] x1 = new double[]{3.5, 2.4, 4.4, 5.6, 1.1};
        double[] x2 = new double[]{3.1, 1.5, -1.5, 2.4, 4.2};

        // Perform forward filtering
        y0 = scaleFilter[0].Encode(x0);
        y1 = scaleFilter[1].Encode(x1);
        y2 = scaleFilter[2].Encode(x2);

        // Display x0
        System.Console.Out.Write("X0 = \{");
        for (int i = 0; i < 4; i++)
            System.Console.Out.Write(x0[i] + ", ");
        // Display summary statistics for X1
        System.Console.Out.Write("X1 Mean: "+scaleFilter[1].Center);
        System.Console.Out.WriteLine("X1 Std. Dev.: "+scaleFilter[1].Spread);
        // Display summary statistics for X2
        System.Console.Out.Write("X2 Median: "+scaleFilter[2].Center);
        System.Console.Out.WriteLine("X2 MAD/0.6745: "+scaleFilter[2].Spread);
        PrintMatrix pm = new PrintMatrix();
    }
}
```
pm.SetTitle("Filtered X0 Using Method=0 (no scaling)"); pm.Print(y0);
pm.SetTitle("Filtered X1 Using Bounded Z-score Scaling\n" +
  "with Center=Mean and Spread=Std. Dev."); pm.Print(y1);
pm.SetTitle("Filtered X2 Using Bounded Z-score Scaling\n" +
  "with Center=Median and Spread=MAD/0.6745"); pm.Print(y2);

// Perform inverse filtering
double[] z0, z1, z2;
z0 = scaleFilter[0].Decode(y0);
z1 = scaleFilter[1].Decode(y1);
z2 = scaleFilter[2].Decode(y2);

Output

X0 = {1.2, 0, -1.4, 1.5, 3.2}
X1 = {3.5, 2.4, 4.4, 5.6, 1.1}
  X1 Mean: 3.4
  X1 Std. Dev.: 1.74212513901843
X2 = {3.1, 1.5, -1.5, 2.4, 4.2}
  X2 Median: 2.4
  X2 MAD/0.6745: 1.33434199665504

Filtered X0 Using Method=0 (no scaling)
  0
  0 1.2
  1 0
  2 -1.4
  3 1.5
  4 3.2

Filtered X1 Using Bounded Z-score Scaling
with Center=Mean and Spread=Std. Dev.
  0 0.0287005788965145
  1 -0.287005788965146
  2 0.287005788965146
  3 0.631412735723321
  4 -0.660113314619835

Filtered X2 Using Bounded Z-score Scaling
with Center=Median and Spread=MAD/0.6745
  0
ScaleFilter.ScalingMethod Enumeration

public enumeration Imsl.DataMining.Neural.ScaleFilter.ScalingMethod

Scaling Method

Fields

**Bounded**

public Imsl.DataMining.Neural.ScaleFilter.ScalingMethod Bounded

Description

Flag to indicate bounded scaling.
**BoundedZScoreMeanStdev**
public Imsl.DataMining.Neural.ScaleFilter.ScalingMethod BoundedZScoreMeanStdev

**Description**
Flag to indicate bounded z-score scaling using the mean and standard deviation.

**BoundedZScoreMedianMAD**
public Imsl.DataMining.Neural.ScaleFilter.ScalingMethod BoundedZScoreMedianMAD

**Description**
Flag to indicate bounded z-score scaling using the median and mean absolute difference.

**None**
public Imsl.DataMining.Neural.ScaleFilter.ScalingMethod None

**Description**
Flag to indicate no scaling.

**UnboundedZScoreMeanStdev**
public Imsl.DataMining.Neural.ScaleFilter.ScalingMethod UnboundedZScoreMeanStdev

**Description**
Flag to indicate unbounded z-score scaling using the mean and standard deviation.

**UnboundedZScoreMedianMAD**
public Imsl.DataMining.Neural.ScaleFilter.ScalingMethod UnboundedZScoreMedianMAD

**Description**
Flag to indicate unbounded z-score scaling using the median and mean absolute difference.

---

**UnsupervisedNominalFilter Class**

public class Imsl.DataMining.Neural.UnsupervisedNominalFilter

Converts nominal data into a series of binary encoded columns for input to a neural network. It also reverses the aforementioned encoding, accepting binary encoded data and returns an array of integers representing the classes for a nominal variable.
Binary Encoding

Method `Encode` (p. 1754) can be used to apply binary encoding. Referring to the result as \( z \), binary encoding takes each category in the nominal variable \( x \), and creates a column in \( z \) containing all zeros and ones. A value of zero indicates that this category was not present and a value of one indicates that it is present.

For example, if \( x[] = \{2, 1, 3, 4, 2, 4\} \) then \( nClasses=4 \), and

\[
\begin{array}{c}
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1
\end{array}
\]

Notice that the number of columns in the result, \( z \), is equal to the number of distinct classes in \( x \). The number of rows in \( z \) is equal to the length of \( x \).

Binary Decoding

Unfiltering can be performed using the method `Decode` (p. 1754). In this case, \( z \) is the input, and we refer to \( x \) as the output. Binary unfiltering takes binary representation in \( z \), and returns the appropriate class in \( x \).

For example, if a row in \( z \) equals \( \{0, 1, 0, 0\} \), then the return value from `Decode` would be 2 for that row. If a row in \( z \) equals \( \{1,0,0,0\} \), then the return value from `Decode` would be 1 for that row. Notice these are the same values as the first two elements of the original \( x \) because classes are numbered sequentially from 1 to \( nClasses \). This ensures that the results of `Decode` are associated with the \( i \)-th class in \( x \).

Property

**NumberOfClasses**

```csharp
virtual public int NumberOfClasses { get; }
```

**Description**
The number of classes in the nominal variable.

**Property Value**
An int containing the number of classes in the nominal variable.

Constructor

**UnsupervisedNominalFilter**

```csharp
public UnsupervisedNominalFilter(int nClasses)
```

Neural Nets UnsupervisedNominalFilter • 1753
Description
Constructor for UnsupervisedNominalFilter.

Parameter
nClasses – An int specifying the number of categories in the nominal variable to be filtered.

Methods

Decode
virtual public int Decode(int[] z)

Description
Decodes a binary encoded array into its nominal category.

Parameter
z – An int array containing the data to be decoded.

Returns
An int containing the number associated with the category encoded in z.

Remarks
This is the inverse of the Encode (p. 1754) method.

Decode
virtual public int[] Decode(int[,] z)

Description
Decodes a matrix representing the binary encoded columns of the nominal variable.

Parameter
z – An int matrix containing the data to be decoded.

Returns
An int array containing the decoded data.

Remarks
This is the inverse of the Encode (p. 1754) method.

Encode
virtual public int[,] Encode(int[] x)

Description
Encodes class data prior to its use in neural network training.

Parameter
x – An int array containing the data to be encoded.
Returns
An int matrix containing the encoded data.

Remarks
Class number must be in the range 1 to nClasses.

Encode
virtual public int[] Encode(int x)

Description
Apply forward encoding to a value.

Parameter
   x – An int containing the value to be encoding.

Returns
An int array containing the encoded data.

Remarks
Class number must be in the range 1 to nClasses.

Example: UnsupervisedNominalFilter
In this example a data set with 7 observations and 3 classes is filtered.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl.DataMining.Neural;

public class UnsupervisedNominalFilterEx1
{
    public static void Main(System.String[] args)
    {
        int nClasses = 3;
        UnsupervisedNominalFilter filter = new UnsupervisedNominalFilter(nClasses);
        int nObs = 7;
        int[] x = new int[]{3, 3, 1, 2, 2, 1, 2};
        int[] xBack = new int[nObs];
        int[,] z;

        // Perform Binary Filtering.
        z = filter.Encode(x);
        PrintMatrix pm = new PrintMatrix();
        pm.SetTitle("Filtered x");
        pm.Print(z);

        // Perform Binary Un-filtering.
        int[] tmp = new int[z.GetLength(1)];
```
for (int i = 0; i < nObs; i++)
{
    for (int j=0; j< z.GetLength(1); j++)
        tmp[j] = z[i,j];
    xBack[i] = filter.Decode(tmp);
}

pm.SetTitle("Result of inverse filtering");
pm.Print(xBack);

Output

Filtered x
0 1 2
0 0 0 1
1 0 0 1
2 1 0 0
3 0 1 0
4 0 1 0
5 1 0 0
6 0 1 0

Result of inverse filtering
0
0 3
1 3
2 1
3 2
4 2
5 1
6 2

UnsupervisedOrdinalFilter Class

public class Imsl.DataMining.Neural.UnsupervisedOrdinalFilter

Encodes ordinal data into percentages for input to a neural network. It also allows decoding, accepting a percentage and converting it into an ordinal value.

Class UnsupervisedOrdinalFilter is designed to either encode or decode ordinal variables. Encoding consists of transforming the ordinal classes into percentages, with each percentage being equal to the percentage of the data at or below this class.

Ordinal Encoding

In this case, x is input to the method Encode (p. 1759) and is filtered by converting each ordinal class value into a cumulative percentage.
For example, if \( x = \{2, 1, 3, 4, 2, 4, 1, 1, 3, 3\} \) then \( n_{Classes} = 4 \), and Encode returns the ordinal class designation with the cumulative percentages displayed in the following table. Cumulative percentages are equal to the percent of the data in this class or a lower class.

<table>
<thead>
<tr>
<th>Ordinal Class</th>
<th>Frequency</th>
<th>Cumulative Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>30%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>50%</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>80%</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100%</td>
</tr>
</tbody>
</table>

Classes in \( x \) must be numbered from 1 to \( n_{Classes} \).

The values returned from encoding or decoding depend upon the setting of Transform (p. 1758). In this example, if the filter was constructed with Transform = TransformMethod.None, then the method Encode will return

\[
z[i] = \{50, 30, 80, 100, 50, 100, 30, 30, 80, 80\}.
\]

If the filter was constructed with Transform = TransformMethod.Sqrt, then the square root of these values is returned; i.e.,

\[
z[i] = \sqrt{\frac{z[i]}{100}}
\]

\[
z[i] = \{0.71, 0.55, 0.89, 1.0, 0.71, 1.0, 0.55, 0.55, 0.89, 0.89\}.
\]

If the filter was constructed with Transform = TransformMethod.AsinSqrt, then the arcsin square root of these values is returned using the following calculation:

\[
z[i] = \arcsin\left(\sqrt{\frac{z[i]}{100}}\right)
\]

### Ordinal Decoding

Ordinal decoding takes a transformed cumulative proportion and converts it into an ordinal class value.

### Properties

**NumberOfClasses**

```csharp
virtual public int NumberOfClasses { get; }
```
Description
The number of categories associated with this ordinal variable.

Property Value
An int containing the number of categories associated with this ordinal variable.

Percentages
virtual public double[] Percentages {get; set; }

Description
The cumulative percentages used during encoding and decoding.

Property Value
A double array of length nClasses containing the cumulative percentages associated with the ordinal categories.

Remarks
If a transform has been applied to the percentages then the transformed percentages are returned. Setting untransformed cumulative percentages with this method bypasses calculating cumulative percentages based on the data being encoded. The percentages must be nondecreasing in the interval [0, 100], with the last element equal to 100. If this method is used it must be called prior to any calls to the encoding and decoding methods.

Transform
virtual public int Transform {get; }

Description
The transform flag used for encoding and decoding.

Property Value
An int containing the transform flag used for encoding and decoding.

Constructor

UnsupervisedOrdinalFilter

Description
Constructor for UnsupervisedOrdinalFilter.

Parameters
- nClasses – An int specifying the number of classes in the data to be filtered.
- transform – An TransformMethod specifying the transform to be applied to the percentages.
Remarks
Values for Transform (p. 1758) are: TransformMethod.None (p. 1762), TransformMethod.Sqrt (p. 1762), TransformMethod.AsinSqrt (p. 1762)

Methods

Decode
virtual public int Decode(double y)

Description
Decodes an encoded ordinal variable.

Parameter
y – A double containing the encoded value to be decoded.

Returns
An int containing the ordinal category associated with y.

Decode
virtual public int[] Decode(double[] y)

Description
Decodes an array of encoded ordinal values.

Parameter
y – A double array containing the encoded ordinal data to be decoded.

Returns
An int array containing the decoded ordinal classifications.

Encode
virtual public double[] Encode(int[] x)

Description
Encodes an array of ordinal categories into an array of transformed percentages.

Parameter
x – An int array containing the categories for the ordinal variable.

Returns
A double array of the transformed percentages.
Remarks
Categories must be numbered from 1 to nClasses.

Encode
virtual public double Encode(int x)

Description
Encodes an ordinal category.

Parameter
x – An int containing the ordinal category.

Returns
A double containing the encoded value, a transformed cumulative percentage.

Remarks
x must be an integer between 1 and nClasses.

Example: UnsupervisedOrdinalFilter
In this example a data set with 10 observations and 4 classes is filtered.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl.DataMining.Neural;

public class UnsupervisedOrdinalFilterEx1
{
    public static void Main(System.String[] args)
    {
        int nClasses = 4;
        UnsupervisedOrdinalFilter filter = new UnsupervisedOrdinalFilter(nClasses,
            UnsupervisedOrdinalFilter.TransformMethod.AsinSqrt);
        int[] x = new int[]{2, 1, 3, 4, 2, 4, 1, 1, 3, 3};
        int nObs = x.Length;
        int[] xBack;
        double[] z;
        // Ordinal Filtering.
        z = filter.Encode(x);
        // Print result without row/column labels.
        PrintMatrix pm = new PrintMatrix();
        PrintMatrixFormat mf;
        mf = new PrintMatrixFormat();
        mf.SetNoRovLabels();
        mf.SetNoColumnLabels();
        pm.SetTitle("Filtered data");
        pm.Print(mf, z);
        // Ordinal Un-filtering.
    }
}
```
pm.SetTitle("Un-filtered data");
xBack = filter.Decode(z);

// Print results of Un-filtering.
pm.Print(mf, xBack);

Output
Filtered data

0.785398163397448
0.579639740363704
1.10714871779409
1.5707963267949
0.785398163397448
1.5707963267949
0.579639740363704
0.579639740363704
1.10714871779409
1.10714871779409

Un-filtered data

2
1
3
4
2
4
1
1
1
3
3

UnsupervisedOrdinalFilter.TransformMethod

Enumeration

public enumeration
Imsl.DataMining.Neural.UnsupervisedOrdinalFilter.TransformMethod

Transform type
Fields

AsinSqrt

Description
Flag to indicate the arcsine square root transform will be applied to the percentages.

None
public Imsl.DataMining.Neural.UnsupervisedOrdinalFilter.TransformMethod None

Description
Flag to indicate no transformation of percentages.

Sqrt

Description
Flag to indicate the square root transform will be applied to the percentages.

TimeSeriesFilter Class

public class Imsl.DataMining.Neural.TimeSeriesFilter

Converts time series data to a lagged format used as input to a neural network.

Class TimeSeriesFilter can be used to operate on a data matrix and lags every column to form a new data matrix. Using the method ComputeLags (p. 1763), each column of the input matrix, x, is transformed into (nLags+1) columns by creating a column for \( \text{lags} = 0, 1, \ldots, \text{nLags} \).

The output data array, \( z \), can be symbolically represented as:

\[
z = [x[0] : x[1] : x[2] : \ldots : x[\text{nLags} - 1]],
\]

where \( x[i] \) is a lagged column of the incoming data matrix, \( x \).

Consider, an example in which \( x \) has five rows and two columns with all variables continuous input attributes. Using \( nObs \) and \( nVar \) to represent the number of rows and columns in \( x \), let

\[
x = \begin{bmatrix}
1 & 6 \\
2 & 7 \\
3 & 8 \\
4 & 9 \\
5 & 10 \\
\end{bmatrix}
\]
If nLags=1, then the number of columns in z[,] is nVar*(nLags+1) = 2*2 = 4, and the number of rows is (nObs-nLags) = 5-1 = 4:

\[
\begin{bmatrix}
1 & 6 & 2 & 7 \\
2 & 7 & 3 & 8 \\
3 & 8 & 4 & 9 \\
4 & 9 & 5 & 10
\end{bmatrix}
\]

If nLags=2, then the number of rows in z will be (nObs-nLags) = (5-2) = 3 and the number of columns will be nVar*(nLags+1) = 2*3 = 6:

\[
\begin{bmatrix}
1 & 6 & 2 & 7 & 3 & 8 \\
2 & 7 & 3 & 8 & 4 & 9 \\
3 & 8 & 4 & 9 & 5 & 10
\end{bmatrix}
\]

### Constructor

**TimeSeriesFilter**

```java
public TimeSeriesFilter()
```

**Description**

Constructor for TimeSeriesClassFilter.

### Method

**ComputeLags**

```java
virtual public double[,] ComputeLags(int nLags, double[,] x)
```

**Description**

Lags time series data to a format used for input to a neural network.

**Parameters**

- `nLags` – An int containing the requested number of lags.
- `x` – A double matrix, nObs by nVar, containing the time series data to be lagged.

**Returns**

A double matrix with (nObs-nLags) rows and (nVar*(nLags+1)) columns. The columns 0 through (nVar-1) contain the columns of x. The next nVar columns contain the first lag of the columns in x, etc.

**Remarks**

nLags must be greater than 0. It is assumed that x is sorted in descending chronological order.
Example: TimeSeriesFilter

In this example a matrix with 5 rows and 2 columns is lagged twice. This produces a two-dimensional matrix with 5 rows, but 2*3=6 columns. The first two columns correspond to lag=0, which just places the original data into these columns. The 3rd and 4th columns contain the first lags of the original 2 columns and the 5th and 6th columns contain the second lags.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl.DataMining.Neural;

public class TimeSeriesFilterEx1
{
    public static void Main(System.String[] args)
    {
        TimeSeriesFilter filter = new TimeSeriesFilter();
        int nLag = 2;
        double[,] x = {{1, 6}, {2, 7}, {3, 8}, {4, 9}, {5, 10}};
        double[,] z = filter.ComputeLags(nLag, x);
        // Print result without row/column labels.
        PrintMatrix pm = new PrintMatrix();
        PrintMatrixFormat mf;
        mf = new PrintMatrixFormat();
        mf.SetNoRowLabels();
        mf.SetNoColumnLabels();
        pm.SetTitle("Lagged data");
        pm.Print(mf, z);
    }
}
```

Output

```
Lagged data
1 6 2 7 3 8
2 7 3 8 4 9
3 8 4 9 5 10
```

TimeSeriesClassFilter Class

public class Imsl.DataMining.Neural.TimeSeriesClassFilter

Converts time series data contained within nominal categories to a lagged format for processing by a neural network. Lagging is done within the nominal categories associated with the time series.
Class `TimeSeriesClassFilter` can be used with a data array, `x`, to compute a new data array, `z[,]`, containing lagged columns of `x`.

When using the method `ComputeLags` (p. 1766), the output array, `z[,]` of lagged columns, can be symbolically represented as:

\[ z = [x[0] : x[1] : x[2] : \ldots : x[nLags - 1]], \]

where `x[i]` is a lagged column of the incoming data array `x`, and `nLags` is the number of computed lags. The lag associated with `x[i]` is equal to the value in `lags[i]`, and lagging is done within the nominal categories given in `iClass`. This requires the time series data in `x[]` be sorted in time order within each category `iClass`.

Consider an example in which the number of observations in `x[]` is 10. There are two lags requested in `lags`. If

\[
\begin{align*}
x^T &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}, \\
iClass^T &= \{1, 1, 1, 1, 1, 1, 1, 1\}, \\
lag^T &= \{0, 2\}
\end{align*}
\]

then, all the time series data fall into a single category, i.e. `nClasses` = 1, and `z` would contain 2 columns and 10 rows. The first column reproduces the values in `x[]` because `lags[0] = 0`, and the second column is the 2nd lag because `lags[1] = 2`.

\[
z = \begin{bmatrix}
1 & 3 \\
2 & 4 \\
3 & 5 \\
4 & 6 \\
5 & 7 \\
6 & 8 \\
7 & 9 \\
8 & NaN \\
9 & NaN \\
10 & NaN
\end{bmatrix}
\]

On the other hand, if the data were organized into two classes with

\[
iClass^T = \{1, 1, 1, 1, 1, 2, 2, 2, 2\},
\]

\[
n == 1765
\]
then `nClasses` is 2, and `z` is still a 2 by 10 matrix, but with the following values:

\[
\begin{pmatrix}
1 & 3 \\
2 & 4 \\
3 & 5 \\
4 & NaN \\
5 & NaN \\
6 & 8 \\
7 & 9 \\
8 & 10 \\
9 & NaN \\
10 & NaN
\end{pmatrix}
\]

The first 5 rows of `z` are the lagged columns for the first category, and the last five are the lagged columns for the second category.

**Constructor**

**TimeSeriesClassFilter**

```csharp
public TimeSeriesClassFilter(int nClasses)
```

*Description*

Constructor for `TimeSeriesClassFilter`.

*Parameter*

`nClasses` – An `int` specifying the number of nominal categories associated with the time series.

**Method**

**ComputeLags**

```csharp
virtual public double[,] ComputeLags(int[] lags, int[] iClass, double[] x)
```

*Description*

Computes lags of an array sorted first by class designations and then descending chronological order.

*Parameters*

`lags` – An `int` array containing the requested lags.

`iClass` – An `int` array containing class number associated with each element of `x`, sorted in ascending order.

`x` – A `double` array containing the time series data to be lagged.
Returns
A double matrix containing the lagged data. The \( i \)-th column of this array is the lagged values of \( x \) for a lag equal to \( \text{lags}[i] \). The number of rows is equal to the length of \( x \).

Remarks
Every lag must be non-negative.
The \( i \)-th element of \( \text{iClass} \) is equal to the class associated with the \( i \)-th element of \( x \). \( \text{iClass} \) and \( x \) must be the same length.
\( x \) is assumed to be sorted first by class designations and then descending chronological order; i.e., most recent observations appear first within a class.

Example: TimeSeriesClassFilter
For illustration purposes, the time series in this example consists of the integers 1, 2, ..., 10, organized into two classes. Of course, it is assumed that these data are sorted in chronologically descending order. That is for each class, the first number is the latest value and the last number in that class is the earliest.
The values 1-4 are in class 1, and the values 5-10 are in class 2. These values represent two separate time series, one for each class. If you were to list them in chronologically ascending order, starting with time=\( T_0 \), the values would be:

- Class 1: \( T_0=4, T_1=3, T_2=2, T_3=1 \)
- Class 2: \( T_0=10, T_1=9, T_2=8, T_3=7, T_4=6, T_5=5 \)

This example requests lag calculations for lags 0, 1, 2, 3. For lag=0, no lagging is performed. For lag=1, the value at time = \( t \) replaced with the value at time = \( t-1 \), the previous value in that class. If \( t-1 < 0 \), then a missing value is placed in that position.
For example, the first lag of a time series at time=\( t \) are the values at time=\( t-1 \). For the time series values of Class 1 (lag=1), these values are:

- Class 1, lag 1: \( T_0=\text{NaN}, T_1=4, T_2=3, T_3=2 \)

The second lag for time=\( t \) consists of the values at time=\( t-2 \):

- Class 1, lag 2: \( T_0=\text{NaN}, T_1=\text{NaN}, T_2=4, T_3=3 \)

Notice that the second lag now has two missing observations. In general, lag=n will have n missing values. In some cases this can result in all missing values for classes with few observations. A class will have all missing values in any of its lag columns that have a lag value larger than or equal to the number of observations in that class.

```csharp
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl.DataMining.Neural;

class TimeSeriesClassFilterEx1
{
    private static int nClasses = 2;
}
```
private static int nObs = 10;
private static int nLags = 4;

public static void Main(System.String[] args)
{
    double[] x = new double[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    double[] time = new double[]{3, 2, 1, 0, 5, 4, 3, 2, 1, 0};
    int[] iClass = new int[]{1, 1, 1, 1, 2, 2, 2, 2, 2, 2};
    int[] lag = new int[]{0, 1, 2, 3};

    // Filter Classified Time Series Data
    TimeSeriesClassFilter filter = new TimeSeriesClassFilter(nClasses);
    double[,] y = filter.ComputeLags(lag, iClass, x);
    double[,] z = new double[nObs, (nLags + 2)];
    // for (int i = 0; i < nObs; i++)
    // {
    //     z[i] = new double[nLags + 2];
    // }
    for (int i = 0; i < nObs; i++)
    {
        z[i,0] = (double) iClass[i];
        z[i,1] = time[i];
        for (int j = 0; j < nLags; j++)
        {
            z[i,j + 2] = y[i,j];
        }
    }

    // Print result without row/column labels.
    PrintMatrix pm = new PrintMatrix();
    PrintMatrixFormat mf;
    mf = new PrintMatrixFormat();
    mf.SetNoRowLabels();
    mf.SetColumnLabels(colLabels);
    pm.SetTitle("Lagged data");
    pm.Print(mf, z);
}

Output

<table>
<thead>
<tr>
<th>Class</th>
<th>Time</th>
<th>Lag=0</th>
<th>Lag=1</th>
<th>Lag=2</th>
<th>Lag=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>NaN</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
Example: Neural Network Application

This application illustrates one common approach to time series prediction using a neural network. In this case, the output target for this network is a single time series. In general, the inputs to this network consist of lagged values of the time series together with other concomitant variables, both continuous and categorical. In this application, however, only the first three lags of the time series are used as network inputs.

The objective is to train a neural network for forecasting the series \( Y_t, t = 0, 1, 2, \ldots \), from the first three lags of \( Y_t \), i.e.

\[
Y_t = f(Y_{t-1}, Y_{t-2}, Y_{t-3})
\]

Since this series consists of data from several company departments, lagging of the series must be done within departments. This creates many missing values. The original data contains 118,519 training patterns. After lagging, 16,507 are identified as missing and are removed, leaving a total of 102,012 usable training patterns. Missing values are denoted using a number not in the training patterns, the value -9,999,999,999.0.

The structure of the network consists of three input nodes and two layers, with three perceptrons in the hidden layer and one in the output layer. The following figure depicts this structure:

![Neural Network Diagram](image-url)

Figure 9. An example 2-layer Feed Forward Neural Network
There are a total of 16 weights in this network, including the 4 bias weights. All perceptrons in the hidden layer use logistic activation, and the output perceptron uses linear activation. Because of the large number of training patterns, the `Activation.LogisticTable` activation function is used instead of `Activation.Logistic`. `Activation.LogisticTable` uses a table lookup for calculating the logistic activation function, which significantly reduces training time. However, these are not completely interchangeable. If a network is trained using `Activation.LogisticTable`, then it is important to use the same activation function for forecasting.

All input nodes are linked to every perceptron in the hidden layer, which are in turn linked to the output perceptron. Then all inputs and the output target are scaled using the `ScaleFilter` class to ensure that all input values and outputs are in the range \([0, 1]\). This requires forecasts to be unscaled using the `Decode()` method of the `ScaleFilter` class.

Training is conducted using the epoch trainer. This trainer allows users to customize training into two stages. Typically this is necessary when training using a large number of training patterns. Stage I training uses randomly selected subsets of training patterns to search for network solutions. Stage II training is optional, and uses the entire set of training patterns. For larger sets of training patterns, training could take many hours, or even days. In that case, Stage II training might be bypassed.

In this example, Stage I training is conducted using the quasi-Newton trainer applied to 20 epochs, each consisting of 5,000 randomly selected observations. Stage II training also uses the quasi-Newton trainer.

The training patterns are contained in two data files: `continuous.txt` and `output.txt`. The formats of these files are identical. The first line of the file contains the number of columns or variables in that file. The second contains a line of tab-delimited integer values. These are the column indices associated with the incoming data. The remaining lines contain tab-delimited, floating point values, one for each of the incoming variables.

For example, the first four lines of the `continuous.txt` file consists of the following lines:

```
3
1 2 3
0 0 0
0 0 0
```

There are 3 continuous input variables which are numbered, or labeled, as 1, 2, and 3.

**Source Code**

```csharp
using System;
using Imsl.DataMining.Neural;
using Imsl.Math;
using System.Runtime.Serialization;

//*****************************************************************************
// NeuralNetworkEx1.java *
// Two Layer Feed-Forward Network Complete Example for Simple Time Series *
//*****************************************************************************

// Synopsis: This example illustrates how to use a Feed-Forward Neural *
// Network to forecast time series data. The network target is a *
// time series and the three inputs are the 1st, 2nd, and 3rd lag *
// for the target series.
```

1770 • TimeSeriesClassFilter  IIMSL C# Numerical Library
// Activation: Logistic_Table in Hidden Layer, Linear in Output Layer *
// Trainer: Epoch Trainer: Stage I - Quasi-Newton, Stage II - Quasi-Newton *
// Inputs: Lags 1-3 of the time series *
// Output: A Time Series sorted chronologically in descending order, *
// i.e., the most recent observations occur before the earliest, *
// within each department *
//*****************************************************************************
// [Serializable]
public class NeuralNetworkEx1 /*: System.Runtime.Serialization.ISerializable */
{
    private static System.String QuasiNewton = "quasi-newton";
    private static System.String LeastSquares = "least-squares";
    // *************************************************************************
    // Network Architecture *
    // *************************************************************************
    private static int nObs = 118519; // number of training patterns
    private static int nInputs = 3; // four inputs
    private static int nContinuous = 3; // one continuous input attribute
    private static int nOutputs = 1; // one continuous output
    private static int nPerceptrons = 3; // perceptrons in hidden layer
    private static int[] perceptrons = new int[] {3}; // # of perceptrons in each
    // hidden layer
    // PERCEPTRON ACTIVATION
    private static IActivation hiddenLayerActivation;
    private static IActivation outputLayerActivation;
    // *************************************************************************
    // Epoch Training Optimization Settings *
    // *************************************************************************
    private static bool trace = true; //trainer logging *
    private static int nEpochs = 20; //number of epochs *
    private static int epochSize = 5000; //samples per epoch *
    // Stage I Trainer - Quasi-Newton Trainer **********************************
    private static int stage1Iterations = 5000; //max. iterations *
    private static double stage1StepTolerance = 1e-09; //step tolerance *
    private static double stage1RelativeTolerance = 1e-11; //rel. tolerance *
    // Stage II Trainer - Quasi-Newton Trainer *********************************
    private static int stage2Iterations = 5000; //max. iterations *
    private static double stage2StepTolerance = 1e-09; //step tolerance *
    private static double stage2RelativeTolerance = 1e-11; //rel. tolerance *
    // *************************************************************************
    // FILE NAMES AND FILE READER DEFINITIONS *
    // *************************************************************************
    // READERS
    private static System.IO.StreamReader contFileInputStream;
    private static System.IO.StreamReader outputFileInputStream;
    // OUTPUT FILES
    // File Name for Serialized Network
    private static System.String networkFileName = "NeuralNetworkEx1.ser";
    // File Name for Serialized Trainer
    private static System.String trainerFileName = "NeuralNetworkTrainerEx1.ser";
    // File Name for Serialized xData File (training input attributes)
    private static System.String xDataFileName = "NeuralNetworkxDataEx1.ser";
    // File Name for Serialized yData File (training output targets)
    private static System.String yDataFileName = "NeuralNetworkyDataEx1.ser";
}

Neural Nets TimeSeriesClassFilter ● 1771
public static void Main(System.String[] args) {
    double[] weight; // Network weights
    double[] gradient; // Network gradient after training
double[,] xData; // Training Patterns Input Attributes
double[,] yData; // Training Targets Output Attributes
double[,] contAtt; // A 2D matrix for the continuous training attributes
double[,] outs; // A matrix containing the training output targets
int i, j, m = 0; // Array indices
int nWeights = 0; // Number of network weights
int nCol = 0; // Number of data columns in input file
int[] ignore; // Array of 0’s and 1’s (0=missing value)
int[] cont_col, outs_col, isMissing = new int[1];
// System.String inputLine = "", temp;
// System.String[] dataElement;
// **********************************************************************
// Initialize timers *
// **********************************************************************
NeuralNetworkEx1.startTime =
DateTime.Now.Hour * 60 * 60 * 1000 +
DateTime.Now.Minute * 60 * 1000 +
DateTime.Now.Second * 1000 +
DateTime.Now.Millisecond;
startTime.ToString());

// ********************************************************************************
// Read continuous attribute data  *
// ********************************************************************************
// Initialize ignore[] for identifying missing observations
ignore = new int[nObs];
isMissing = new int[1];
openInputFiles();

nCol = readFirstLine(contFileInputStream);
nContinuous = nCol;
System.Console.Out.WriteLine("--> Number of continuous variables: " +
nContinuous);
// If the number of continuous variables is greater than zero then read
// the remainder of this file (contFile)
if (nContinuous > 0)
{
    // contFile contains continuous attribute data
    contAtt = new double[nObs, nContinuous];
    double[] _contAttRow = new double[nContinuous];
    // for (int i2 = 0; i2 < nObs; i2++)
    // {
    //     contAtt[i2] = new double[nContinuous];
    // }
    cont_col = readColumnLabels(contFileInputStream, nContinuous);
    for (i = 0; i < nObs; i++)
    {
        isMissing[0] = -1;
        _contAttRow = readDataLine(contFileInputStream, nContinuous,
                                   isMissing);
        for (int jj=0; jj < nContinuous; jj++)
        {
            contAtt[i,jj] = _contAttRow[jj];
        }
    }
ignore[i] = isMissing[0];
if (isMissing[0] >= 0)
    m++;
}
}
else  
{
    nContinuous = 0;
    contAtt = new double[1,1];
    // for (int i3 = 0; i3 < 1; i3++)
    // {
    //     contAtt[i3] = new double[1];
    // }
    contAtt[0,0] = 0;
}
closeFile(contFileInputStream);
// ************************************************************************
// Read continuous output targets
// ************************************************************************
ncol = readFirstLine(outputFileInputStream);
nOutputs = ncol;
System.Console.Out.WriteLine("--> Number of output variables: ");
outs = new double[nObs, nOutputs];
double[] _outsRow = new double[nOutputs];
// for (int i4 = 0; i4 < nObs; i4++)
// {
//     outs[i4] = new double[nOutputs];
// }
// Read numeric labels for continuous input attributes
outs_col = readColumnLabels(outputFileInputStream, nOutputs);

m = 0;
for (i = 0; i < nObs; i++)
{
    isMissing[0] = ignore[i];
    _outsRow = readDataLine(outputFileInputStream, nOutputs, isMissing);
    for (int jj = 0; jj < nOutputs; jj++)
    {
        outs[i, jj] = _outsRow[jj];
    }
    ignore[i] = isMissing[0];
    if (isMissing[0] >= 0)
        m++;
}
closeFile(outputFileInputStream);
// Remove missing observations using the ignore[] array
m = removeMissingData(nObs, nContinuous, ignore, contAtt);
m = removeMissingData(nObs, nOutputs, ignore, outs);
System.Console.Out.WriteLine("--> Total Number of Training Patterns: ");
nObs = nObs - m;
System.Console.Out.WriteLine("--> Number of Usable Training Patterns: ");
+ nObs;

// Setup Method and Bounds for Scale Filter
ScaleFilter scaleFilter = new ScaleFilter(
    ScaleFilter.ScalingMethod.Bounded);
scaleFilter.SetBounds(lowerDataLimit, upperDataLimit, 0, 1);

// PREPROCESS TRAINING PATTERNS

System.Console.Out.WriteLine(
    "--> Starting Preprocessing of Training Patterns");
xData = new double[nObs, nContinuous];
// for (int i5 = 0; i5 < nObs; i5++)
// { xData[i5] = new double[nContinuous];
// }
yData = new double[nObs, nOutputs];
// for (int i6 = 0; i6 < nObs; i6++)
// { yData[i6] = new double[nOutputs];
// }
for (i = 0; i < nObs; i++)
{
    for (j = 0; j < nContinuous; j++)
    { xData[i,j] = contAtt[i,j];
    }
    yData[i,0] = outs[i,0];
}
scaleFilter.Encode(0, xData);
scaleFilter.Encode(1, xData);
scaleFilter.Encode(2, xData);
scaleFilter.Encode(0, yData);

// CREATE FEEDFORWARD NETWORK

System.Console.Out.WriteLine("--> Creating Feed Forward Network Object");
FeedForwardNetwork network = new FeedForwardNetwork();
// setup input layer with number of inputs = nInputs = 3
network.InputLayer.CreateInputs(nInputs);
// create a hidden layer with nPerceptrons=3 perceptrons
network.CreateHiddenLayer().CreatePerceptrons(nPerceptrons);
// create output layer with nOutputs=1 output perceptron
network.OutputLayer.CreatePerceptrons(nOutputs);
// link all inputs and perceptrons to all perceptrons in the next layer
network.LinkAll();
// Get Network Perceptrons for Setting Their Activation Functions
Perceptron[] perceptrons = network.Perceptrons;
// Set all hidden layer perceptrons to logistic_table activation
for (i = 0; i < perceptrons.Length - 1; i++)
{ perceptrons[i].Activation = hiddenLayerActivation;
}
perceptrons[perceptrons.Length - 1].Activation = outputLayerActivation;
System.Console.Out.WriteLine("--> Feed Forward Network Created with 2 Layers");
// **********************************************************************
// TRAIN NETWORK USING EPOCH TRAINER *
// **********************************************************************
ITrainer trainer = createTrainer(QuasiNewton, QuasiNewton);
startTime =
    DateTime.Now.Hour * 60 * 60 * 1000 +
    DateTime.Now.Minute * 60 * 1000 +
    DateTime.Now.Second * 1000 +
    DateTime.Now.Millisecond;
// Train Network
trainer.Train(network, xData, yData);

// Check Training Error Status
switch (trainer.ErrorStatus)
{
    case 0: errorMsg = errorMsg0; break;
    case 1: errorMsg = errorMsg1; break;
    case 2: errorMsg = errorMsg2; break;
    case 3: errorMsg = errorMsg3; break;
    case 4: errorMsg = errorMsg4; break;
    case 5: errorMsg = errorMsg5; break;
    case 6: errorMsg = errorMsg6; break;
    default: errorMsg = "--> Unknown Error Status Returned from Trainer"; break;
}
System.Console.Out.WriteLine(errorMsg);
int currentTimeNow =
    DateTime.Now.Hour * 60 * 60 * 1000 +
    DateTime.Now.Minute * 60 * 1000 +
    DateTime.Now.Second * 1000 +
    DateTime.Now.Millisecond;
    currentTimeNow.ToString());
double duration = (double) (currentTimeNow - startTime) / 1000.0;
    " seconds");
double[] stats = network.ComputeStatistics(xData, yData); // Display Network Errors
*************************************************************************");
System.Console.Out.WriteLine("--- SSE: " + (float)stats[0]);
System.Console.Out.WriteLine("--- RMS: " + (float)stats[1]);
System.Console.Out.WriteLine("--- Largest Absolute Residual: " + (float)stats[4]);
System.Console.Out.WriteLine("*************************************************************************");
System.Console.Out.WriteLine("\n");

// **********************************************************************
// OBTAIN AND DISPLAY NETWORK WEIGHTS AND GRADIENTS
// **********************************************************************
System.Console.Out.WriteLine("--- Getting Network Weights and Gradients\n");
weight = network.Weights;
// Get number of weights = number of gradients
nWeights = network.NumberOfWeights;
// Obtain Gradient Vector
gradient = trainer.ErrorGradient;
// Print Network Weights and Gradients
System.Console.Out.WriteLine("\n");
System.Console.Out.WriteLine("--- Network Weights and Gradients:\n");
System.Console.Out.WriteLine("*************************************************************************");
double[,] printMatrix = new double[nWeights,2];
for (int i7 = 0; i7 < nWeights; i7++)
{
    printMatrix[i7] = new double[2];
}
for (i = 0; i < nWeights; i++)
{
    printMatrix[i,0] = weight[i];
    printMatrix[i,1] = gradient[i];
}
// Print result without row/column labels.
System.String[] colLabels = new System.String[]{"Weight", "Gradient"};
PrintMatrix pm = new PrintMatrix();
PrintMatrixFormat mf;
mf = new PrintMatrixFormat();
mf.SetNoRowLabels();
mf.SetColumnLabels(colLabels);
matrix.SetTitle("Weights and Gradients");
mf.Print(mf, printMatrix);
System.Console.Out.WriteLine("\n--> Saving Trained Network into " + 
networkFileName);
write(network, networkFileName);
System.Console.Out.WriteLine("--> Saving Network Trainer into " + 
trainerFileName);
write(trainer, trainerFileName);
System.Console.Out.WriteLine("--> Saving xData into " + xDataFileName);
write(xData, xDataFileName);
System.Console.Out.WriteLine("--> Saving yData into " + yDataFileName);
write(yData, yDataFileName);
}
// *************************************************************************
// OPEN DATA FILES
// *************************************************************************
static public void openInputFiles()
{
    try
    {
        // Continuous Input Attributes
        System.IO.Stream contInputStream = new System.IO.FileStream( 
            contFileName, System.IO.FileMode.Open, System.IO.FileAccess.Read);
        contFileInputStream = new System.IO.StreamReader(new
            System.IO.StreamReader(contInputStream).BaseStream, 
            System.Text.Encoding.UTF7);
        // Continuous Output Targets
        System.IO.Stream outputInputStream = new System.IO.FileStream( 
            outputFileName, System.IO.FileMode.Open, System.IO.FileAccess.Read);
        outputFileInputStream = new System.IO.StreamReader( 
            new System.IO.StreamReader(outputInputStream).BaseStream, 
            System.Text.Encoding.UTF7);
    }
    catch (System.Exception e)
    {
        System.Console.Out.WriteLine("-->ERROR: " + e);
        System.Environment.Exit(0);
    }
} //*************************************************************************
// READ FIRST LINE OF DATA FILE AND RETURN NUMBER OF COLUMNS IN FILE
// *************************************************************************
static public int readFirstLine(System.IO.StreamReader inputFile)
{
    System.String inputLine = "", temp;
    int nCol = 0;
    try
    {
        temp = inputFile.ReadLine();
        inputLine = temp.Trim();
        nCol = System.Int32.Parse(inputLine);
    }
    catch (System.Exception e)
    {
        System.Console.Out.WriteLine("-->ERROR: " + e);
        System.Environment.Exit(0);
    }
}
System.Console.Out.WriteLine("--> ERROR READING 1st LINE OF File" + e);
System.Environment.Exit(0);
}

return nCol;
}

// *************************************************************************
// READ COLUMN LABELS (2ND LINE IN FILE) *
// *************************************************************************
static public int[] readColumnLabels(System.IO.StreamReader inputFile,
int nCol)
{
int[] contCol = new int[nCol];
System.String inputLine = "", temp;
System.String[] dataElement;
// Read numeric labels for continuous input attributes
try
{
    temp = inputFile.ReadLine();
    inputLine = temp.Trim();
}
catch (System.Exception e)
{
        + e);
    System.Environment.Exit(0);
}
dataElement = inputLine.Split(new Char[] { ' '});
for (int i = 0; i < nCol; i++)
{
    contCol[i] = System.Int32.Parse(dataElement[i]);
}
return contCol;
}

// *************************************************************************
// READ DATA ROW *
// *************************************************************************
static public double[] readDataLine(System.IO.StreamReader inputFile,
int nCol, int[] isMissing)
{
    double missingValueIndicator = -9999999999.0;
    double[] dataLine = new double[nCol];
    double[] contCol = new double[nCol];
    System.String inputLine = "", temp;
    System.String[] dataElement;
    try
    {
        temp = inputFile.ReadLine();
        inputLine = temp.Trim();
    }
    catch (System.Exception e)
    {
        System.Console.Out.WriteLine("-->ERROR READING LINE: " + e);
        System.Environment.Exit(0);
    }
dataElement = inputLine.Split(new Char[] { ' '});
for (int j = 0; j < nCol; j++)
dataLine[j] = System.Double.Parse(dataElement[j]);
if (dataLine[j] == missingValueIndicator)
   isMissing[0] = 1;
}
return dataLine;
}

// ***************************************************************
// CLOSE FILE
// ***************************************************************
static public void closeFile(System.IO.StreamReader inputFile)
{
try
{
   inputFile.Close();
}
catch (System.Exception e)
{
   System.Console.Out.WriteLine("ERROR: Unable to close file: " + e);
   System.Environment.Exit(0);
}
}

// ***************************************************************
// REMOVE MISSING DATA
// ***************************************************************
// Now remove all missing data using the ignore[] array
// and recalculate the number of usable observations, nObs
// This method is inefficient, but it works. It removes one case at a
// time, starting from the bottom. As a case (row) is removed, the cases
// below are pushed up to take it's place.
// ***************************************************************
static public int removeMissingData(int nObs, int nCol, int[] ignore,
double[,] inputArray)
{
   int m = 0;
   for (int i = nObs - 1; i >= 0; i--)
   {
      if (ignore[i] >= 0)
      {
         // the ith row contains a missing value
         // remove the ith row by shifting all rows below the
         // ith row up by one position, e.g. row i+1 \rightarrow row i
         m++;
         if (nCol > 0)
         {
            for (int j = i; j < nObs - m; j++)
            {
               for (int k = 0; k < nCol; k++)
               {
                  inputArray[j,k] = inputArray[j + 1,k];
               }
            }
         }
      }
   }
   return m;
}
static public ITrainer createTrainer(System.String s1, System.String s2) {
    EpochTrainer epoch = null; // Epoch Trainer (returned by this method)
    QuasiNewtonTrainer stage1Trainer; // Stage I Quasi-Newton Trainer
    QuasiNewtonTrainer stage2Trainer; // Stage II Quasi-Newton Trainer
    LeastSquaresTrainer stage1LS; // Stage I Least Squares Trainer
    LeastSquaresTrainer stage2LS; // Stage II Least Squares Trainer
    int currentTimeNow; // Calendar time tracker

    // Create Epoch (Stage I/Stage II) trainer from above trainers.
    if (s1.Equals(QuasiNewton))
    {
        // Setup stage I quasi-newton trainer
        stage1Trainer = new QuasiNewtonTrainer();
        stage1Trainer.MaximumTrainingIterations = stage1Iterations;
        stage1Trainer.StepTolerance = stage1StepTolerance;
        if (s2.Equals(QuasiNewton))
        {
            stage2Trainer = new QuasiNewtonTrainer();
            stage2Trainer.MaximumTrainingIterations = stage2Iterations;
            epoch = new EpochTrainer(stage1Trainer, stage2Trainer);
        }
        else
        {
            if (s2.Equals(LeastSquares))
            {
                stage2LS = new LeastSquaresTrainer();
                stage2LS.InitialTrustRegion = 1.0e-3;
                stage2LS.MaximumTrainingIterations = stage2Iterations;
                epoch = new EpochTrainer(stage1Trainer, stage2LS);
            }
            else
            {
                epoch = new EpochTrainer(stage1Trainer);
            }
        }
    }
    else
    {
        // Setup stage I least squares trainer
        stage1LS = new LeastSquaresTrainer();
        stage1LS.InitialTrustRegion = 1.0e-3;
        stage1LS.MaximumTrainingIterations = stage1Iterations;
        if (s2.Equals(QuasiNewton))
        {
            stage2Trainer = new QuasiNewtonTrainer();
            stage2Trainer.MaximumStepsize(maxStepSize);
        }
    }
}
stage2Trainer.MaximumTrainingIterations = stage2Iterations;
epoch = new EpochTrainer(stage1LS, stage2Trainer);
}
else
{
if (s2.Equals(LeastSquares))
{
stage2LS = new LeastSquaresTrainer();
stage2LS.InitialTrustRegion = 1.0e-3;
//stage2LS.setMaximumStepsize(maxStepSize);
stage2LS.MaximumTrainingIterations = stage2Iterations;
epoch = new EpochTrainer(stage1LS, stage2LS);
}
else
{
epoch = new EpochTrainer(stage1LS);
}
}
epoch.NumberOfEpochs = nEpochs;
epoch.EpochSize = epochSize;
epoch.Random = new Imsl.Stat.Random(1234567);
epoch.SetRandomSamples(new Imsl.Stat.Random(12345),
new Imsl.Stat.Random(67891));
System.Console.Out.WriteLine(" --> Trainer: Stage I - " + s1 +
" Stage II " + s2);
System.Console.Out.WriteLine(" --> Number of Epochs: " + nEpochs);
System.Console.Out.WriteLine(" --> Epoch Size: " + epochSize);
// Describe optimization setup for Stage I training
System.Console.Out.WriteLine(" --> Creating Stage I Trainer");
System.Console.Out.WriteLine(" --> Stage I Iterations: " +
stage1Iterations);
stage1StepTolerance);
stage1RelativeTolerance);
System.Console.Out.WriteLine(" --> Stage I Step Size: " +
"DEFAULT");
System.Console.Out.WriteLine(" --> Stage I Trace: " +
trace);
if (s2.Equals(QuasiNewton) || s2.Equals(LeastSquares))
{
// Describe optimization setup for Stage II training
System.Console.Out.WriteLine(" --> Creating Stage II Trainer");
System.Console.Out.WriteLine(" --> Stage II Iterations: " +
stage2Iterations);
System.Console.Out.WriteLine(" --> Stage II Step Tolerance: " +
stage2StepTolerance);
System.Console.Out.WriteLine(" --> Stage II Relative Tolerance: " +
stage2RelativeTolerance);
System.Console.Out.WriteLine(" --> Stage II Step Size: " +
"DEFAULT");
System.Console.Out.WriteLine(" --> Stage II Trace: " +
trace);
}
currentTimeNow =
Output

--> Starting Data Preprocessing at: 44821683
--> Number of continuous variables: 3
--> Number of output variables: 1
--> Number of Missing Observations: 16507
--> Total Number of Training Patterns: 118519
--> Number of Usable Training Patterns: 102012
--> Starting Preprocessing of Training Patterns
--> Creating Feed Forward Network Object
--> Feed Forward Network Created with 2 Layers
--> Training Network using Epoch Trainer
   --> Creating Epoch Trainer
   --> Trainer: Stage I - quasi-newton Stage II quasi-newton
   --> Number of Epochs: 20
   --> Epoch Size: 5000
   --> Creating Stage I Trainer
   --> Stage I Iterations: 5000
   --> Stage I Step Tolerance: 1E-09
   --> Stage I Relative Tolerance: 1E-11
   --> Stage I Step Size: DEFAULT
   --> Stage I Trace: True
   --> Creating Stage II Trainer
   --> Stage II Iterations: 5000
   --> Stage II Step Tolerance: 1E-09
Starting Network Training at 45070408

The last global step failed to locate a lower point than the current error value. The current solution may be an approximate solution and no more accuracy is possible, or the step tolerance may be too large.

Network Training Completed at: 52311842
Training Time: 7241.434 seconds

SSE: 4.49772
RMS: 0.1423779
Laplacian Error: 103.4631
Scaled Laplacian Error: 0.1707173
Largest Absolute Residual: 0.4921748

Getting Network Weights and Gradients

Weights and Gradients:

-248.425149158357  -9.50818419128144E-05
-4.01301691047852  -9.08459022567118E-07
248.602873209042   -2.84623837579401E-05
258.622104579914   -8.49451049786515E-05
0.125785905718184  -7.51083204612989E-07
-258.811023180184  -2.81816574426092E-05
-4.256726621727131  -5.25467092773031E-07
394.428311058654   -2.76079812235378E-05
422.855887848789   -1.40339989032276E-06
-1.01024996891467  -8.54119524733673E-07
422.854960914701   -3.37315953950562E-08
91.0301749864326   -0.000555459860183764
0.672279284955327   -3.11957565142863E-06
-91.0431760187523  -0.00012020875079491
-422.186774012951  -1.36689037615365E-06

Saving Trained Network into NeuralNetworkEx1.ser
Saving Network Trainer into NeuralNetworkTrainerEx1.ser
Saving xData into NeuralNetworkxDataEx1.ser
Saving yData into NeuralNetworkyDataEx1.ser

Results

The above output indicates that the network successfully completed its training. The final sum of squared errors was 3.88, and the RMS (the scaled version of the sum of squared errors) was 0.12. All of the gradients at this solution are nearly zero, which is expected if network training found a local or global optima. Non-zero gradients usually indicate there was a problem with network training.
--- Starting Data Preprocessing at: 6385241
--- Number of continuous variables: 3
--- Number of output variables: 1
--- Number of Missing Observations: 16507
--- Total Number of Training Patterns: 118519
--- Number of Usable Training Patterns: 102012
--- Starting Preprocessing of Training Patterns
--- Creating Feed Forward Network Object
--- Feed Forward Network Created with 2 Layers
--- Training Network using Epoch Trainer
  --- Creating Epoch Trainer
  --- Trainer: Stage I - quasi-newton Stage II quasi-newton
  --- Number of Epochs: 20
  --- Epoch Size: 5000
--- Creating Stage I Trainer
  --- Stage I Iterations: 5000
  --- Stage I Step Tolerance: 1E-09
  --- Stage I Relative Tolerance: 1E-11
  --- Stage I Step Size: DEFAULT
  --- Stage I Trace: True
--- Creating Stage II Trainer
  --- Stage II Iterations: 5000
  --- Stage II Step Tolerance: 1E-09
  --- Stage II Relative Tolerance: 1E-11
  --- Stage II Step Size: DEFAULT
  --- Stage II Trace: True
--- Starting Network Training at 6396512
--- The last global step failed to locate a lower point than the current error value. The current solution may be an approximate solution and no more accuracy is possible, or the step tolerance may be too large.
--- Network Training Completed at: 7070947
--- Training Time: 674.435 seconds
******************************************************************************
--- SSE: 4.39503
--- RMS: 0.1391272
--- Laplacian Error: 115.7665
--- Scaled Laplacian Error: 0.1910183
--- Largest Absolute Residual: 0.510379
******************************************************************************
--- Getting Network Weights and Gradients
--- Network Weights and Gradients:
******************************************************************************

<table>
<thead>
<tr>
<th>Weight</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.49062717385272</td>
<td>-7.35781305464078E-07</td>
</tr>
<tr>
<td>-6.59528808872046</td>
<td>4.80847787053182E-05</td>
</tr>
<tr>
<td>4401.51680674985</td>
<td>-1.85429409555815E-08</td>
</tr>
<tr>
<td>0.704338219368497</td>
<td>-8.32580200754921E-07</td>
</tr>
<tr>
<td>-0.725623033199978</td>
<td>5.70308208281333E-05</td>
</tr>
<tr>
<td>2805.39692819902</td>
<td>-1.44534276811825E-08</td>
</tr>
<tr>
<td>-3.93198626371718</td>
<td>-2.13822710907307E-06</td>
</tr>
<tr>
<td>4.04824502335685</td>
<td>6.44781336574968E-05</td>
</tr>
<tr>
<td>-1787.37723908109</td>
<td>-1.7603927485418E-09</td>
</tr>
</tbody>
</table>

Neural Nets  TimeSeriesClassFilter ● 1785
38.3945130188417   -4.36277172935969E-05
37.636974274678   -2.57936093318423E-05
-0.0378671989334698   7.43008547201402E-07
0.4447658217999   -0.000427699567365199
-0.495282381760965   -0.000323163187876674
-1037.62096685008   1.57344076332679E-08
-37.6465155277179   -7.36849253095175E-05

***********************************************

--> Saving Trained Network into NeuralNetworkEx1.ser
--> Saving Network Trainer into NeuralNetworkTrainerEx1.ser
--> Saving xData into NeuralNetworkxDataEx1.ser
--> Saving yData into NeuralNetworkyDataEx1.ser

Links to Input Data Files Used in this Example and the Training Log:

Network Class

public class Imsl.DataMining.Neural.Network
Neural network base class.

Properties

InputLayer
abstract public Imsl.DataMining.Neural.InputLayer InputLayer {get; }

Description
The InputLayer object.

Property Value
The Network InputLayer.

Links
abstract public Imsl.DataMining.Neural.Link[] Links {get; }

Description
An array containing the Link objects in the Network.
Property Value

An array of Links associated with this Network.

**NumberOfInputs**

abstract public int NumberOfInputs {get; }

**Description**

The number of Network inputs.

**Property Value**

An int which contains the number of inputs.

**NumberOfLinks**

abstract public int NumberOfLinks {get; }

**Description**

The number of Network Links among the Nodes.

**Property Value**

An int which contains the number of Links (p. 1665) in the Network.

**NumberOfOutputs**

abstract public int NumberOfOutputs {get; }

**Description**

The number of Network output Perceptrons (p. 1661).

**Property Value**

An int which contains the number of outputs.

**NumberOfWeights**

abstract public int NumberOfWeights {get; }

**Description**

The number of Weights (p. 1666) in the Network.

**Property Value**

An int which contains the number of Weights associated with this Network.

**OutputLayer**

abstract public Imsl.DataMining.Neural.OutputLayer OutputLayer {get; }

**Description**

The OutputLayer.

**Property Value**

The Network OutputLayer.

**Perceptrons**

abstract public Imsl.DataMining.Neural.Perceptron[] Perceptrons {get; }
**Description**
An array containing the Perceptrons in the Network.

**Property Value**
An array of Perceptrons associated with this Network.

**Weights**
abstract public double[] Weights {get; set; }

**Description**
The Weights (p. 1666).

**Property Value**
A double array containing the Weights associated with Network Links (p. 1665).

**Constructor**

**Network**
public Network()

**Description**
Default constructor for Network.

**Remarks**
Since this class is abstract, it cannot be instantiated directly; this constructor is used by constructors in classes derived from Network.

**Methods**

**ComputeStatistics**
virtual public double[] ComputeStatistics(double[,] xData, double[,] yData)

**Description**
Computes error statistics.

**Parameters**
- `xData` – A double matrix containing the input values.
- `yData` – A double array containing the observed values.

**Returns**
A double array containing the above described statistics.
Remarks
This is a static method that can be used to compute the statistics regardless of the training class used to train the Network.

Computes statistics related to the error. In this table, the observed values are $y_i$. The forecasted values are $\hat{y}_i$. The mean observed value is $\bar{y} = \sum_i y_i / N$, where $N$ is the number of observations and $C$ is the number of classes per observation.

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SSE</td>
<td>$\frac{1}{2} \sum_i (y_i - \hat{y}_i)^2$</td>
</tr>
<tr>
<td>1</td>
<td>RMS</td>
<td>$\frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$</td>
</tr>
<tr>
<td>2</td>
<td>Laplacian</td>
<td>$\sum_i</td>
</tr>
<tr>
<td>3</td>
<td>Scaled Laplacian</td>
<td>$\frac{\sum_i</td>
</tr>
<tr>
<td>4</td>
<td>Max residual</td>
<td>$\max_i</td>
</tr>
</tbody>
</table>

CreateHiddenLayer
abstract public Imsl.DataMining.Neural.HiddenLayer CreateHiddenLayer()

Description
Creates the next HiddenLayer in the Network.

Returns
The new HiddenLayer.

Forecast
abstract public double[] Forecast(double[] x)

Description
Returns a forecast for each of the Network’s outputs computed from the trained Network.

Parameter
    x – A double array of values with the same length and order as the training patterns used to train the Network.

Returns
A double array containing the forecasts for the output Perceptrons (p. 1661). Its length is equal to the number of output Perceptrons.

GetForecastGradient
abstract public double[,] GetForecastGradient(double[] x)

Description
Returns the derivatives of the outputs with respect to the Weights (p. 1666) evaluated at x.

Parameter
    x – A double array which specifies the input values at which the gradient is to be evaluated.
Returns
A double array containing the gradient values. The value of gradient[i][j] is \( dy_i/dw_j \), where \( y_i \) is the \( i \)-th output and \( w_j \) is the \( j \)-th weight.

Example: Network
This example uses a network previously trained and serialized into four files to obtain information about the network and forecasts. Training was done using the code for the FeedForwardNetwork Example 1.

The network training targets were generated using the relationship:
\[
y = 10 \times X_1 + 20 \times X_2 + 30 \times X_3 + 2.0 \times X_4,
\]
where \( X_1 \) to \( X_3 \) are the three binary columns, corresponding to categories 1 to 3 of the nominal attribute, and \( X_4 \) is the scaled continuous attribute.

The structure of the network consists of four input nodes and two layers, with three perceptrons in the hidden layer and one in the output layer. The following figure illustrates this structure:

![Figure 10. An example 2-layer Feed Forward Neural Network with 4 Inputs](image)

All perceptrons were trained using a Linear Activation Function. Forecasts are generated for 9 conditions, corresponding to the following conditions:
- Nominal Class 1-3 with the Continuous Input Attribute = 0
- Nominal Class 1-3 with the Continuous Input Attribute = 5.0
- Nominal Class 1-3 with the Continuous Input Attribute = 10.0
Note that the network training statistics retrieved from the serialized network confirm that this is the same network used in the previous example. Obtaining these statistics requires retrieval of the training patterns which were serialized and stored into separate files. This information is not serialized with the network, nor with the trainer.

using System;
using Imsl.DataMining.Neural;
using System.Runtime.Serialization;
//*****************************************************************************
// Two Layer Feed-Forward Network with 4 inputs: 1 categorical with 3 classes
// encoded using binary encoding and 1 continuous input, and 1 output
// target (continuous). There is a perfect linear relationship between
// the input and output variables:
// MODEL: Y = 10*X1 + 20*X2 + 30*X3 + 2*X4
// Variables X1-X3 are the binary encoded nominal variable and X4 is the
// continuous variable.
// This example uses Linear Activation in both the hidden and output layers
// The network uses a 2-layer configuration, one hidden layer and one
// output layer. The hidden layer consists of 3 perceptrons. The output
// layer consists of a single output perceptron.
// The input from the continuous variable is scaled to [0,1] before training
// the network. Training is done using the Quasi-Newton Trainer.
// The network has a total of 19 weights.
// Since the network target is a linear combination of the network inputs, and
// since all perceptrons use linear activation, the network is able to forecast
// the every training target exactly. The largest residual is 2.78E-08.
//*****************************************************************************
[Serializable]
public class NetworkEx1
{
    // **********************************************************************
    // MAIN
    // **********************************************************************

    public static void Main(System.String[] args)
    {
        double[,] xData; // Input Attributes for Training Patterns
        double[,] yData; // Output Attributes for Training Patterns
        double[] weight; // network weights
        double[] gradient; // network gradient after training
        // Input Attributes for Forecasting
        double[,] x = {{1, 0, 0, 0.0}, {0, 1, 0, 0.0}, {0, 0, 1, 0.0},
                       {1, 0, 0, 5.0}, {0, 1, 0, 5.0}, {0, 0, 1, 5.0},
                       {1, 0, 0, 10.0}, {0, 1, 0, 10.0}, {0, 0, 1, 10.0}};
        double[] xTemp, y; // Temporary areas for storing forecasts
        int i, j; // loop counters
        // Names of Serialized Files
        System.String networkFileName = "FeedForwardNetworkEx1.ser"; // the network
        System.String trainerFileName = "FeedForwardTrainerEx1.ser"; // the trainer
    }
}
System.String xDataFileName = "FeedForwardxDataEx1.ser"; // xData
System.String yDataFileName = "FeedForwardyDataEx1.ser"; // yData
// **********************************************************************
// READ THE TRAINED NETWORK FROM THE SERIALIZED NETWORK OBJECT
// **********************************************************************
System.Console.Out.WriteLine("---> Reading Trained Network from " +
networkFileName);
Network network = (Network) read(networkFileName);
// **********************************************************************
// READ THE SERIALIZED XDATA[,] AND YDATA[,] ARRAYS OF TRAINING
// PATTERNS.
// **********************************************************************
System.Console.Out.WriteLine("---> Reading xData from " + xDataFileName);
xData = (double[,]) read(xDataFileName);
System.Console.Out.WriteLine("---> Reading yData from " + yDataFileName);
yData = (double[,]) read(yDataFileName);
// **********************************************************************
// READ THE SERIALIZED TRAINER OBJECT
// **********************************************************************
System.Console.Out.WriteLine("---> Reading Network Trainer from " +
trainerFileName);
ITrainer trainer = (ITrainer) read(trainerFileName);
// **********************************************************************
// DISPLAY TRAINING STATISTICS
// **********************************************************************
double[] stats = network.ComputeStatistics(xData, yData);
// Display Network Errors
System.Console.Out.WriteLine("***********************************************");
System.Console.Out.WriteLine("---> SSE: " +
(float)stats[0]);
System.Console.Out.WriteLine("---> RMS: " +
(float)stats[1]);
(float)stats[2]);
(float)stats[3]);
(float)stats[4]);
System.Console.Out.WriteLine("***********************************************");
System.Console.Out.WriteLine(" ");
for (i = 0; i < nWeights; i++)
{...}
System.Console.Out.WriteLine("w[" + i + "]=" + (float) weight[i] + " g[" + i + "]=" + (float) gradient[i]);

// **********************************************************************
// OBTAIN AND DISPLAY FORECASTS FOR THE LAST 10 TRAINING TARGETS
// **********************************************************************
// Get number of network inputs
int nInputs = network.NumberOfInputs;
// Get number of network outputs
int nOutputs = network.NumberOfOutputs;
xTemp = new double[nInputs]; // temporary x space for forecast inputs
y = new double[nOutputs]; // temporary y space for forecast output
System.Console.Out.WriteLine(" ");
// Obtain example forecasts for input attributes = x[]
// X1-X3 are binary encoded for one nominal variable with 3 classes
// X4 is a continuous input attribute ranging from 0-10. During
// training, X4 was scaled to [0,1] by dividing by 10.
for (i = 0; i < 9; i++)
{
    for (j = 0; j < nInputs; j++)
        xTemp[j] = x[i,j];
xTemp[nInputs - 1] = xTemp[nInputs - 1] / 10.0;
y = network.Forecast(xTemp);
System.Console.Out.WriteLine(" --\r\n--> X1=" + (int) x[i,0] + " X2=" + (int) x[i,1] + " X3=" + (int) x[i,2] + " | X4=" + x[i,3]);
System.Console.Out.WriteLine(" | \r\n    y=" + (float) (10.0 * x[i,0] + 20.0 * x[i,1] + 30.0 * x[i,2] + 2.0 * x[i,3]) + " | Forecast=" + (float) y[0]);
}

// **************************************************************************
// READ Serialized NETWORK FROM A FILE
// **************************************************************************
static public System.Object read(System.String filename)
{
    System.IO.FileStream fis = new System.IO.FileStream(filename,
        System.IO.FileMode.Open, System.IO.FileAccess.Read);
    IFormatter ois = new BinaryFormatter();
    System.Object obj = (System.Object) ois.Deserialize(fis);
    fis.Close();
    return obj;
}

// ***************************************************
// ** Neural Networks **
// ***************************************************

Output
-- Reading Trained Network from FeedForwardNetworkEx1.ser
-- Reading xData from FeedForwardxDataEx1.ser
-- Reading yData from FeedForwardyDataEx1.ser
-- Reading Network Trainer from FeedForwardTrainerEx1.ser
*********************************************************
-- SSE: 1.013444E-15
-- RMS: 2.007463E-19
-- Laplacian Error: 3.005804E-07

Neural Nets Network • 1793
---> Scaled Laplacian Error: 3.535235E-10
---> Largest Absolute Residual: 2.784275E-08
***************************************************

--> Getting Network Information

--> Network Weights and Gradients:
\[ w[0] = -1.491785, g[0] = -2.611079E-08 \]
\[ w[1] = -1.491785, g[1] = -2.611079E-08 \]
\[ w[2] = -1.491785, g[2] = -2.611079E-08 \]
\[ w[7] = 4.725622, g[7] = -5.273856E-08 \]
\[ w[8] = 4.725622, g[8] = -5.273856E-08 \]
\[ w[10] = 6.217407, g[10] = -8.733E-10 \]
\[ w[12] = 1.072258, g[12] = -1.690978E-07 \]
\[ w[14] = 1.072258, g[14] = -1.690978E-07 \]
\[ w[15] = 3.850755, g[15] = -1.7029E-08 \]
\[ w[16] = 3.850755, g[16] = -1.7029E-08 \]
\[ w[17] = 3.850755, g[17] = -1.7029E-08 \]
\[ w[18] = 2.411725, g[18] = -1.588144E-08 \]

---> X1=1 X2=0 X3=0 | X4=0 | y=10 | Forecast=10
---> X1=0 X2=1 X3=0 | X4=0 | y=20 | Forecast=20
---> X1=0 X2=0 X3=1 | X4=0 | y=30 | Forecast=30
---> X1=1 X2=0 X3=0 | X4=5 | y=20 | Forecast=20
---> X1=0 X2=1 X3=0 | X4=5 | y=30 | Forecast=30
---> X1=0 X2=0 X3=1 | X4=5 | y=40 | Forecast=40
---> X1=1 X2=0 X3=0 | X4=10 | y=30 | Forecast=30
---> X1=0 X2=1 X3=0 | X4=10 | y=40 | Forecast=40
---> X1=0 X2=0 X3=1 | X4=10 | y=50 | Forecast=50
Chapter 28: Error Handling

Types

class Warning ................................................................. 1795
class WarningObject ....................................................... 1797
class IMSLException ....................................................... 1798
class Logger ................................................................. 1799

Warning Class

public class Imsl.Warning
Handles warning messages.
This class maintains a single, private, WarningObject that actually displays the warning messages.

Properties

WarningObject
static public Imsl.WarningObject WarningObject {get; set; }

Description
The WarningObject allows warning errors to be handled in a more custom fashion.

Property Value
The current WarningObject.

Remarks
WarningObject may be set to null, in which case error messages will be ignored.
Writer
static public System.IO.TextWriter Writer {get; set; }

Description
The stream to which warning messages are to be written.

Remarks
The input may be null, in which case warnings are not written.

Constructor

Warning
public Warning()

Description
Initializes a new instance of the Imsl.Warning (p. 1795) class.

Method

Print
static public void Print(object source, string bundleName, string key, object[] arg)

Description
Issues a warning message.

Parameters
  source – The Object that is the source of the warning.
  bundleName – A String which specifies the base name of the resource. The actual name is
    formed by appending ".ErrorMessages".
  key – A String which specifies the warning message in the resource.
  arg – A Object which specifies arguments used to format the message.

Remarks
Warning messages are stored as MessageFormat patterns in a ResourceBundle. This method retrieves
the pattern from the bundle, formats the message with the supplied arguments, and prints the message to the
warning stream.
WarningObject Class

public class Imsl.WarningObject
Handles warning messages.

Property

Writer
public System.IO.TextWriter Writer {get; set; }

Description
Reassigns the writer.

Remarks
The new warning writer may be set to null, in which case warnings are not printed.

Constructor

WarningObject
public WarningObject()

Description
Handle warning messages.

Method

Print
virtual public void Print(object source, string baseName, string key, object[] arg)

Description
Issue a warning message.

Parameters

source – The Object that is the source of the warning.
baseName – A String which specifies the base name of the resource. The actual name is formed by appending ".ErrorMessages".

Error Handling
key – A String which specifies the warning message in the resource.
arg – A Object which specifies arguments used to format the message.

Remarks
Warning messages are stored as string format items in a resource. This method retrieves the format from
the resource, formats the message with the supplied arguments, and prints the message to the warning
stream.

IMSLException Class

public class Imsl.IMSLException : ApplicationException : ISerializable
Signals that a mathematical exception has occurred.

Constructors
IMSLException
IMSLException()
Description
Constructs an IMSLException with no detail message.
Remarks
A detail message is a String that describes this particular exception.

IMSLException
IMSLException(string s)
Description
Constructs an IMSLException with the specified detail message.
Parameter
s – A String which specifies the detail message.
Remarks
A detail message is a String that describes this particular exception.

IMSLException
IMSLException(string namespaceName, string key, object[] arguments)
Description
Constructs an IMSLException with the specified detail message.
Parameters

namespaceName – A String which specifies the namespace containing the ErrorMessages resource bundle.

direction – A String which specifies the key of the error message in the resource bundle.

Arguments – An array of Objects containing arguments used within the error message string.

Remarks

The error message String is in a resource bundle, ErrorMessages.

IMSLException

IMSLException(string message, System.Exception exception)

Description

Constructs an IMSLException with the specified detail message.

Parameters

message – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

IMSLException

IMSLException(System.Runtime.Serialization.SerializationInfo info,

Description

Constructs an IMSLException with the serialized data.

Parameters

info – The Object that holds the serialized object data.

c context – The contextual information about the source or destination.

Logger Class

public class Imsl.Logger

This class is used to implement logging in some IMSL algorithms. Its design mimics the Java java.util.Logging class, but is simpler, meeting only the requirements of logging intermediate results and notes from IMSL classes.
Properties

LogLevel
public Imsl.Logger.Level LogLevel {get; set; }

Description
The Logger.Level associated with this Logger. The finer grained levels provide more detailed output and are inclusive of the levels before it. Not all levels are implemented for all classes.

Writer
public System.IO.TextWriter Writer {get; set; }

Description

Constructor

Logger
public Logger()

Description
Create a new Logger object using the default TextWriter (System.Console.Out) and the default level (Level.Fine)

Methods

Log
public void Log(Imsl.Logger.Level logLevel, string className, string methodName, string message)

Description
Log a message.

Parameters

logLevel – The Logger.Level of the message. If the logger is set to a lower log level, the message is not logged.
className – The name of the class issuing the log message.
methodName – The name of the method issuing the log message.
message – The string message to log.
Remarks
The output of a successful log entry is of the form “ClassName: message”

Log
public void Log(Imsl.Logger.Level logLevel, string className, string
methodName, string message, object[] arg)

Description
Log a message with parameters.

Parameters
logLevel – The Logger.Level of the message. If the logger is set to a lower log level, the
message is not logged.
className – The name of the class issuing the log message.
methodName – The name of the method issuing the log message.
message – The string message to log that contains placeholders of the form {0} to contain each
element in the parameter list.
arg – An Object array of parameter values.

Remarks
The output of a successful log entry is of the form “ClassName: formatted message”

Log
public void Log(Imsl.Logger.Level logLevel, string className, string
methodName, string name, string key, object[] arg)

Description
Log a message with parameters using a resource file.

Parameters
logLevel – The Logger.Level of the message. If the logger is set to a lower log level, the
message is not logged.
className – The name of the class issuing the log message.
methodName – The name of the method issuing the log message.
name – The name of the resource bundle within the IMSL C# Library.
key – The key for the message string in the resource bundle.
arg – An Object array of parameter values.

Remarks
The output of a successful log entry is of the form “ClassName: formatted message”
Chapter 29: Exceptions

Types

class IMSLUnexpectedErrorException .............................................. 1806
class AllConstraintsNotSatisfiedException ................................. 1806
class BadInitialGuessException .................................................. 1807
class BoundaryInconsistentException ....................................... 1808
class BoundsInconsistentException .......................................... 1810
class ConstraintEvaluationException ....................................... 1811
class ConstraintsInconsistentException .................................. 1812
class ConstraintsNotSatisfiedException .................................... 1814
class CorrectorConvergenceException ..................................... 1815
class CyclingOccurringException ........................................... 1816
class DidNotConvergeException .............................................. 1817
class EqualityConstraintsException ........................................ 1819
class ErrorTestException ......................................................... 1820
class FalseConvergenceException ........................................... 1822
class IllConditionedException .................................................. 1823
class InconsistentSystemException .......................................... 1824
class InitialConstraintsException ............................................ 1825
class InvalidMPSFileException .................................................. 1827
class IterationMatrixSingularException ................................... 1828
class LimitingAccuracyException ............................................. 1829
class LinearlyDependentGradientsException ............................... 1830
class MaxIterationsException .................................................. 1831
class MaxNumberStepsAllowedException .................................. 1833
class MaxFcnEvalsExceededException ...................................... 1834
class MultipleSolutionsException ............................................ 1835
class NoAcceptablePivotException .......................................... 1836
class NoAcceptableStepsizeException ..................................... 1837
class NoConvergenceException ............................................... 1839
class NoLPSolutionException ................................................... 1840
class NoProgressException ..................................................... 1841
class NotDefiniteAMatrixException .......................................... 1842
class IllConditionedException .......................................................... 1899
class IncreaseErrRelException ......................................................... 1900
class InitialMAException ............................................................... 1902
class InvalidMatrixException .......................................................... 1903
class InvalidPartialCorrelationException ........................................ 1903
class MatrixSingularException ......................................................... 1904
class MoreObsDelThanEnteredException .......................................... 1905
class NegativeFreqException ........................................................... 1906
class NegativeWeightException ......................................................... 1908
class NewInitialGuessException ....................................................... 1909
class NoAcceptableModelFoundException ........................................ 1910
class NoConvergenceException ........................................................ 1911
class NoDegreesOfFreedomException .............................................. 1913
class NoProgressException ............................................................. 1914
class NoVariationInputException ...................................................... 1932
class NoVectorXException ............................................................... 1934
class NonInvertibleException .......................................................... 1917
class NonPosVarianceException ....................................................... 1918
class NonPosVarianceXYException ................................................... 1920
class NonPositiveEigenvalueException ............................................ 1921
class NonStationaryException ........................................................ 1922
class NoPositiveVarianceException ............................................... 1923
class NotCDFException ................................................................. 1925
class NotPositiveDefiniteException ............................................... 1927
class NotPositiveSemiDefiniteException ......................................... 1928
class NotSemiDefiniteException ...................................................... 1929
class NoVariablesEnteredException ............................................... 1930
class NoVariablesException .......................................................... 1931
class NoVariationInputException ..................................................... 1932
class NoVectorXException .............................................................. 1934
class PooledCovarianceSingularException ....................................... 1935
class RankException ................................................................. 1936
class RankDeficientException ....................................................... 1937
class ScaleFactorZeroException ...................................................... 1938
class SingularException ............................................................... 1939
class SingularTriangularMatrixException ...................................... 1940
class SumOfWeightsNegException .................................................. 1941
class TooManyCallsException ......................................................... 1942
class TooManyFunctionEvaluationsException .................................. 1944
class TooManyIterationsException ............................................... 1945
class TooManyIterationsRetryException ....................................... 1946
class TooManyJacobianEvalException ......................................... 1948
class TooManyObsDeletedException .............................................. 1949
class VarsDeterminedException ...................................................... 1950
class ZeroNormException .............................................................. 1951

class ZeroNormException .............................................................. 1952
IMSLUnexpectedErrorException Class

public class Imsl.IMSLUnexpectedErrorException : IMSLException : ISerializable
Signals that an unexpected error has occurred.
Please contact your IMSL technical support representative to report this problem.

Constructor

IMSLUnexpectedErrorException
public IMSLUnexpectedErrorException()

Description
Constructs an IMSLUnexpectedErrorException.

AllConstraintsNotSatisfiedException Class

public class Imsl.Math.AllConstraintsNotSatisfiedException : IMSLException : ISerializable
All constraints are not satisfied.
L1 minimization was applied to all constraints (including bounds and simple variables) except the equalities to approximate violated non-equalities as well as possible. If a feasible solution is possible then try using refinement.

Constructors

AllConstraintsNotSatisfiedException
public AllConstraintsNotSatisfiedException()

Description
All constraints are not satisfied.

AllConstraintsNotSatisfiedException
public AllConstraintsNotSatisfiedException(string message)
**Description**

All constraints are not satisfied.

**Parameter**

- `message` – The error message that explains the reason for the exception.

### AllConstraintsNotSatisfiedException

```java
public AllConstraintsNotSatisfiedException(string message, System.Exception exception)
```

**Description**

All constraints are not satisfied.

**Parameters**

- `message` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

### AllConstraintsNotSatisfiedException

```java
```

**Description**

All constraints are not satisfied.

**Parameters**

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**BadInitialGuessException Class**

```java
public class Imsl.Math.BadInitialGuessException : IMSLException : ISerializable
```

Penalty function point infeasible for original problem. Try new initial guess.

### Constructors

**BadInitialGuessException**

```java
public BadInitialGuessException()
```

---

**Exceptions**

| BadInitialGuessException | 1807 |
**Description**
Penalty function point infeasible for original problem. Try new initial guess.

**BadInitialGuessException**
```csharp
public BadInitialGuessException(string message)
```

**Description**
Penalty function point infeasible for original problem. Try new initial guess.

**Parameter**
- `message` – The error message that explains the reason for the exception.

**BadInitialGuessException**
```csharp
public BadInitialGuessException(string message, System.Exception exception)
```

**Description**
Penalty function point infeasible for original problem. Try new initial guess.

**Parameters**
- `message` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**BadInitialGuessException**
```csharp
BadInitialGuessException(System.Runtime.Serialization.SerializationInfo info,
```

**Description**
Penalty function point infeasible for original problem. Try new initial guess.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**BoundaryInconsistentException Class**

```csharp
public class Imsl.Math.BoundaryInconsistentException : IMSLException : ISerializable
```

The boundary conditions are inconsistent.
Constructors

**BoundaryInconsistentException**

```java
public BoundaryInconsistentException(string side, string gridVal)
```

**Description**

The boundary conditions are inconsistent.

**Parameters**

- **side** – A String, indicates if the left or right boundary conditions are inconsistent.
- **gridVal** – A String, the grid value for which the boundary conditions are inconsistent, i.e. “xMin” for the left and “xMax” for the right boundaries.

**BoundaryInconsistentException**

```java
public BoundaryInconsistentException()
```

**Description**

The boundary conditions are inconsistent.

**BoundaryInconsistentException**

```java
public BoundaryInconsistentException(string s, object[] args)
```

**Description**

The boundary conditions are inconsistent.

**Parameters**

- **s** – The error message that explains the reason for the exception.
- **args** – Arguments for the error message.

**BoundaryInconsistentException**

```java
public BoundaryInconsistentException(string message)
```

**Description**

The boundary conditions are inconsistent.

**Parameter**

- **message** – The error message that explains the reason for the exception.

**BoundaryInconsistentException**

```java
public BoundaryInconsistentException(string s, System.Exception exception)
```

**Description**

The boundary conditions are inconsistent.
Parameters

- The error message that explains the reason for the exception.
- The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

BoundaryInconsistentException


Description

The boundary conditions are inconsistent.

Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

BoundsInconsistentException Class

public class Imsl.Math.BoundsInconsistentException : IMSLException : ISerializable

The bounds given are inconsistent.

Constructors

BoundsInconsistentException

public BoundsInconsistentException(string nameVariable, string nameLowerBound, string nameUpperBound, int index, double lowerBound, double upperBound)

Description

The bounds given are inconsistent.

Parameters

- nameVariable – Name of the variable being bounded.
- nameLowerBound – Name of the lower bound.
- nameUpperBound – Name of the upper bound.
- index – The index of the inconsistent bound.
- lowerBound – Value of the lower bound.
upperBound – Value of the upper bound.

**BoundsInconsistentException**

public BoundsInconsistentException()

*Description*
The bounds given are inconsistent.

**BoundsInconsistentException**

public BoundsInconsistentException(string message)

*Description*
The bounds given are inconsistent.

*Parameter*

message – The error message that explains the reason for the exception.

**BoundsInconsistentException**

public BoundsInconsistentException(string s, System.Exception exception)

*Description*
The bounds given are inconsistent.

*Parameters*

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**BoundsInconsistentException**


*Description*
The bounds given are inconsistent.

*Parameters*

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

**ConstraintEvaluationException Class**

public class Imsl.Math.ConstraintEvaluationException : IMSLException : ISerializable

Constraint evaluation returns an error with current point.

**Exceptions**

ConstraintEvaluationException • 1811
Constructors

ConstraintEvaluationException
public ConstraintEvaluationException()

Description
Constraint evaluation returns an error with current point.

ConstraintEvaluationException
public ConstraintEvaluationException(string message)

Description
Constraint evaluation returns an error with current point.

Parameter
message – The error message that explains the reason for the exception.

ConstraintEvaluationException
public ConstraintEvaluationException(string s, System.Exception exception)

Description
Constraint evaluation returns an error with current point.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

ConstraintEvaluationException

Description
Constraint evaluation returns an error with current point.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

ConstraintsInconsistentException Class

public class Imsl.Math.ConstraintsInconsistentException : IMSLException : ISerializable

1812 • ConstraintsInconsistentException  IMSL C# Numerical Library
The equality constraints are inconsistent.

**Constructors**

```csharp
public ConstraintsInconsistentException()

Description
The equality constraints are inconsistent.
```

```csharp
public ConstraintsInconsistentException(string s, object[] args)

Description
The constraints are inconsistent.

Parameters
- **s** – The error message that explains the reason for the exception.
- **args** – Arguments for the error message.
```

```csharp
public ConstraintsInconsistentException(string message)

Description
The equality constraints are inconsistent.

Parameter
- **message** – The error message that explains the reason for the exception.
```

```csharp
public ConstraintsInconsistentException(string s, System.Exception exception)

Description
The equality constraints are inconsistent.

Parameters
- **s** – The error message that explains the reason for the exception.
- **exception** – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.
```

```csharp
```

**Exceptions**

- `ConstraintsInconsistentException` • 1813
The equality constraints are inconsistent.

Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**ConstraintsNotSatisfiedException Class**

```csharp
public class Imsl.Math.ConstraintsNotSatisfiedException : IMSLError : ICloneable

No vector x satisfies all of the constraints.
```

**Constructors**

**ConstraintsNotSatisfiedException**

```csharp
public ConstraintsNotSatisfiedException()
```

**Description**

No vector x satisfies all of the constraints.

**ConstraintsNotSatisfiedException**

```csharp
public ConstraintsNotSatisfiedException(string message)
```

**Description**

No vector x satisfies all of the constraints.

**Parameter**

- message – The error message that explains the reason for the exception.

**ConstraintsNotSatisfiedException**

```csharp
public ConstraintsNotSatisfiedException(string s, System.Exception exception)
```

**Description**

No vector x satisfies all of the constraints.

**Parameters**

- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.
ConstraintsNotSatisfiedException


Description
No vector $x$ satisfies all of the constraints.

Parameters
- **info** – The object that holds the serialized object data.
- **context** – The contextual information about the source or destination.

CorrectorConvergenceException Class

public class Imsl.Math.CorrectorConvergenceException : IMSLException : ISerializable
Corrector failed to converge.

Constructors

CorrectorConvergenceException

public CorrectorConvergenceException(double time)

Description
Corrector failed to converge.

Parameter
- **time** – A double scalar, the time point at which the convergence exception occured.

CorrectorConvergenceException

public CorrectorConvergenceException()

Description
Corrector failed to converge.

CorrectorConvergenceException

public CorrectorConvergenceException(string s, object[] args)

Description
Corrector failed to converge.

Exceptions

CorrectorConvergenceException • 1815
Parameters

- `s` – The error message that explains the reason for the exception.
- `args` – Arguments for the error message.

**CorrectorConvergenceException**

```csharp
public CorrectorConvergenceException(string message)
```

Description

Corrector failed to converge.

Parameter

- `message` – The error message that explains the reason for the exception.

**CorrectorConvergenceException**

```csharp
public CorrectorConvergenceException(string s, System.Exception exception)
```

Description

Corrector failed to converge.

Parameters

- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**CorrectorConvergenceException**

```csharp
CorrectorConvergenceException(System.Runtime.Serialization.SerializationInfo info,
```

Description

Corrector failed to converge.

Parameters

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

**CyclingOccurringException Class**

```csharp
public class Imsl.Math.CyclingOccurringException : IMSLException : ISerializable
```

The algorithm appears to be cycling. Using refinement may help.
Constructors

**CyclingOccurringException**

public CyclingOccurringException()

**Description**
The algorithm appears to be cycling. Using refinement may help.

**CyclingOccurringException**

public CyclingOccurringException(string message)

**Description**
The algorithm appears to be cycling. Using refinement may help.

**Parameter**
message – The error message that explains the reason for the exception.

**CyclingOccurringException**

public CyclingOccurringException(string message, System.Exception exception)

**Description**
The algorithm appears to be cycling. Using refinement may help.

**Parameters**
message – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**CyclingOccurringException**


**Description**
The algorithm appears to be cycling. Using refinement may help.

**Parameters**
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

---

**DidNotConvergeException Class**

public class Imsl.Math.DidNotConvergeException : IMSLException : ISerializable

Exceptions

DidNotConvergeException • 1817
Constructors

**DidNotConvergeException**

```java
public DidNotConvergeException()
```

**Description**

Maximum number of iterations exceeded.

**DidNotConvergeException**

```java
public DidNotConvergeException(int maximumNumberOfIterations)
```

**Description**

Maximum number of iterations exceeded.

**Parameter**

- `maximumNumberOfIterations` – Maximum number of iterations allowed exceeded argument.

**DidNotConvergeException**

```java
public DidNotConvergeException(int info, int min)
```

**Description**

Maximum number of iterations exceeded.

**Parameters**

- `info` – First argument for SVD.DidNotConverge string.
- `min` – Second argument for SVD.DidNotConverge string.

**DidNotConvergeException**

```java
public DidNotConvergeException(string s, object[] args)
```

**Description**

Maximum function evaluations exceeded.

**Parameters**

- `s` – The error message that explains the reason for the exception.
- `args` – An `Object` array which specifies arguments used to format the message.

**DidNotConvergeException**

```java
public DidNotConvergeException(string message)
```

**Description**

Maximum number of iterations exceeded.
DidNotConvergeException

public DidNotConvergeException(string message, int maximumNumberOfIterations)

Description
Maximum number of iterations exceeded.

Parameters
- message – The error message that explains the reason for the exception.
- maximumNumberOfIterations – Maximum number of iterations allowed.

DidNotConvergeException

public DidNotConvergeException(string s, System.Exception exception)

Description
Maximum number of iterations exceeded.

Parameters
- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

DidNotConvergeException


Description
Maximum number of iterations exceeded.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

EqualityConstraintsException Class

public class Imsl.Math.EqualityConstraintsException : IMSLException : ISerializable

The variables are determined by the equality constraints.

Exceptions

EqualityConstraintsException • 1819
Constructors

**EqualityConstraintsException**

```
public EqualityConstraintsException()
```

*Description*
The variables are determined by the equality constraints.

```
public EqualityConstraintsException(string message)
```

*Description*
The variables are determined by the equality constraints.

*Parameter*

- **message** – The error message that explains the reason for the exception.

```
public EqualityConstraintsException(string s, System.Exception exception)
```

*Description*
The variables are determined by the equality constraints.

*Parameters*

- **s** – The error message that explains the reason for the exception.
- **exception** – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```
```

*Description*
The variables are determined by the equality constraints.

*Parameters*

- **info** – The object that holds the serialized object data.
- **context** – The contextual information about the source or destination.

---

**ErrorTestException Class**

```
public class Imsl.Math.ErrorTestException : IMSLException : ISerializable
```

Error test failure detected.

---

1820 • ErrorTestException

IMSL C# Numerical Library
Constructors

**ErrorTestException**

```java
public ErrorTestException(double time, double stepsize)
```

**Description**

Error test failure detected.

**Parameters**

- `time` – A double scalar, the time point at which the error test failure occurred.
- `stepsize` – A double scalar, the step size used in the integration.

**ErrorTestException**

```java
public ErrorTestException()
```

**Description**

Error test failure detected.

**ErrorTestException**

```java
public ErrorTestException(string s, object[] args)
```

**Description**

Error test failure detected.

**Parameters**

- `s` – The error message that explains the reason for the exception.
- `args` – Arguments for the error message.

**ErrorTestException**

```java
public ErrorTestException(string message)
```

**Description**

Error test failure detected.

**Parameter**

- `message` – The error message that explains the reason for the exception.

**ErrorTestException**

```java
public ErrorTestException(string s, System.Exception exception)
```

**Description**

Error test failure detected.

Exceptions

- **ErrorTestException** • 1821
Parameters

s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

ErrorTestException

Description
Error test failure detected.

Parameters

info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

FalseConvergenceException Class

public class Imsl.Math.FalseConvergenceException : IMSLException : ISerializable

False convergence, the iterates appear to be converging to a noncritical point.

Constructors

FalseConvergenceException
public FalseConvergenceException()

Description
False convergence, the iterates appear to be converging to a noncritical point.

FalseConvergenceException
public FalseConvergenceException(string message)

Description
False convergence, the iterates appear to be converging to a noncritical point.
Parameter

message – The error message that explains the reason for the exception.

FalseConvergenceException

public FalseConvergenceException(string s, System.Exception exception)

Description
False convergence, the iterates appear to be converging to a noncritical point.

Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

FalseConvergenceException


Description
False convergence, the iterates appear to be converging to a noncritical point.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

IllConditionedException Class

public class Imsl.Math.IllConditionedException : IMSLEException : ISerializable

Problem is singular or ill-conditioned.

Constructors

IllConditionedException

public IllConditionedException()
Description
Problem is singular or ill-conditioned.

IllConditionedException
public IllConditionedException(string message)

Description
Problem is singular or ill-conditioned.

Parameter
message – The error message that explains the reason for the exception.

IllConditionedException
public IllConditionedException(string s, System.Exception exception)

Description
Problem is singular or ill-conditioned.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException
parameter is not a null reference, the current exception is raised in a catch block that handles the
inner exception.

IllConditionedException
IllConditionedException(System.Runtime.Serialization.SerializationInfo info,

Description
Problem is singular or ill-conditioned.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

InconsistentSystemException Class

public class Imsl.Math.InconsistentSystemException : IMSLErrException :
ISerializable
Inconsistent system.
Constructors

**InconsistentSystemException**
public InconsistentSystemException()

**Description**
Inconsistent system.

**InconsistentSystemException**
public InconsistentSystemException(string message)

**Description**
Inconsistent system.

**Parameter**
- message – The error message that explains the reason for the exception.

**InconsistentSystemException**
public InconsistentSystemException(string s, System.Exception exception)

**Description**
Inconsistent system.

**Parameters**
- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**InconsistentSystemException**

**Description**
Inconsistent system.

**Parameters**
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**InitialConstraintsException Class**

public class Imsl.Math.InitialConstraintsException : IMSLException : ISerializable

| Exceptions          | InitialConstraintsException • 1825 |
The constraints at the initial point are inconsistent.

** Constructors **

**InitialConstraintsException**

public InitialConstraintsException()

** Description **
The constraints at the initial point are inconsistent.

**InitialConstraintsException**

public InitialConstraintsException(string message)

** Description **
The constraints at the initial point are inconsistent.

** Parameter **

message – The error message that explains the reason for the exception.

**InitialConstraintsException**

public InitialConstraintsException(string s, object[] args)

** Description **
The constraints at the initial point are inconsistent.

** Parameters **

s – The error message that explains the reason for the exception.

args – Arguments for the error message.

**InitialConstraintsException**

public InitialConstraintsException(string s, System.Exception exception)

** Description **
The constraints at the initial point are inconsistent.

** Parameters **

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**InitialConstraintsException**

Description
The constraints at the initial point are inconsistent.

Parameters
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

InvalidMPSFileException Class

```csharp
public class Imsl.Math.InvalidMPSFileException : IMSLException : ISerializable
Invalid MPS file.
```

Constructors

```csharp
InvalidMPSFileException
public InvalidMPSFileException(string message)
```

Description
Invalid MPS file.

Parameter
- `message` – The error message that explains the reason for the exception.

```csharp
InvalidMPSFileException
public InvalidMPSFileException(string s, object[] args)
```

Description
Invalid MPS file.

Parameters
- `s` – The error message that explains the reason for the exception.
- `args` – Arguments for the error message.

```csharp
InvalidMPSFileException
InvalidMPSFileException(System.Runtime.Serialization.SerializationInfo info,
```

Description
Invalid MPS file.

Exceptions InvalidMPSFileException • 1827
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**IterationMatrixSingularException Class**

```csharp
public class Imsl.Math.IterationMatrixSingularException : IMSLException : ISerializable
{
    Iteration matrix is singular.
}
```

**Constructors**

**IterationMatrixSingularException**

```csharp
public IterationMatrixSingularException(double time, double stepsize)
{
    Iteration matrix is singular.
}
```

**Parameters**

- time – A double scalar, the time point at which the iteration matrix is singular.
- stepsize – A double scalar, the stepsize used in the integration.

**IterationMatrixSingularException**

```csharp
public IterationMatrixSingularException()
{
    Iteration matrix is singular.
}
```

**IterationMatrixSingularException**

```csharp
public IterationMatrixSingularException(string s, object[] args)
{
    Iteration matrix is singular.
}
```

**Parameters**

- s – The error message that explains the reason for the exception.
- args – Arguments for the error message.

**IterationMatrixSingularException**

```csharp
public IterationMatrixSingularException(string message)
{
    Iteration matrix is singular.
}
```
Description
Iteration matrix is singular.

Parameter
message – The error message that explains the reason for the exception.

IterationMatrixSingularException
public IterationMatrixSingularException(string s, System.Exception exception)

Description
Iteration matrix is singular.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

IterationMatrixSingularException

Description
Iteration matrix is singular.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

LimitingAccuracyException Class

public class Imsl.Math.LimitingAccuracyException : IMSLException : ISerializable
Limiting accuracy reached for a singular problem.

Constructors

LimitingAccuracyException
public LimitingAccuracyException()

Exceptions LimitingAccuracyException • 1829
**LimitingAccuracyException**

public LimitingAccuracyException(string message)

**Description**

Limiting accuracy reached for a singular problem.

**Parameter**

message – The error message that explains the reason for the exception.

**LimitingAccuracyException**

public LimitingAccuracyException(string s, System.Exception exception)

**Description**

Limiting accuracy reached for a singular problem.

**Parameters**

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**LimitingAccuracyException**


**Description**

Limiting accuracy reached for a singular problem.

**Parameters**

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

---

**LinearlyDependentGradientsException Class**

public class Imsl.Math.LinearlyDependentGradientsException : IMSLException : ISerializable

Working set gradients are linearly dependent.
Constructors

**LinearlyDependentGradientsException**
public LinearlyDependentGradientsException()

**Description**
Working set gradients are linearly dependent.

**LinearlyDependentGradientsException**
public LinearlyDependentGradientsException(string message)

**Description**
Working set gradients are linearly dependent.

**Parameter**
message – The error message that explains the reason for the exception.

**LinearlyDependentGradientsException**
public LinearlyDependentGradientsException(string s, System.Exception exception)

**Description**
Working set gradients are linearly dependent.

**Parameters**
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**LinearlyDependentGradientsException**

**Description**
Working set gradients are linearly dependent.

**Parameters**
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

---

**MaxIterationsException Class**

public class Imsl.Math.MaxIterationsException : IMSLException : ISerializable

---

Exceptions

MaxIterationsException • 1831
Maximum number of iterations exceeded.

Constructors

MaxIterationsException
public MaxIterationsException()

Description
Maximum number of iterations exceeded.

MaxIterationsException
public MaxIterationsException(string message)

Description
Maximum number of iterations exceeded.

Parameter
message – The error message that explains the reason for the exception.

MaxIterationsException
public MaxIterationsException(string s, System.Exception exception)

Description
Maximum number of iterations exceeded.

Parameters
s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

MaxIterationsException

Description
Maximum number of iterations exceeded.

Parameters
info – The object that holds the serialized object data.

context – The contextual information about the source or destination.
MaxNumberStepsAllowedException Class

```csharp
public class Imsl.Math.MaxNumberStepsAllowedException : IMSLException : ISerializable

Maximum number of steps allowed exceeded.
```

Constructors

**MaxNumberStepsAllowedException**
```csharp
public MaxNumberStepsAllowedException()
```

**Description**
Maximum number of steps allowed exceeded.

**Parameter**

```csharp
maxSteps – Maximum number of steps allowed.
```

**MaxNumberStepsAllowedException**
```csharp
public MaxNumberStepsAllowedException(int maxSteps)
```

**Description**
Maximum number of steps allowed exceeded.

**Parameter**

```csharp
maxSteps – Maximum number of steps allowed.
```

**MaxNumberStepsAllowedException**
```csharp
public MaxNumberStepsAllowedException(string message)
```

**Description**
Maximum number of steps allowed exceeded.

**Parameter**

```csharp
message – The error message that explains the reason for the exception.
```

**MaxNumberStepsAllowedException**
```csharp
public MaxNumberStepsAllowedException(string s, System.Exception exception)
```

**Description**
Maximum number of steps allowed exceeded.

**Parameters**

```csharp
s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException
parameter is not a null reference, the current exception is raised in a catch block that handles the
inner exception.
```

Exceptions

**MaxNumberStepsAllowedException** • 1833
MaxNumberStepsAllowedException

Description
Maximum number of steps allowed exceeded.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

MaxFcnEvalsExceededException Class

public class Imsl.Math.MaxFcnEvalsExceededException : IMSLException : ISerializable
Maximum function evaluations exceeded.

Constructors

MaxFcnEvalsExceededException
public MaxFcnEvalsExceededException()

Description
Maximum function evaluations exceeded.

MaxFcnEvalsExceededException
public MaxFcnEvalsExceededException(string message)

Description
Maximum function evaluations exceeded.

Parameter
- message – The error message that explains the reason for the exception.

MaxFcnEvalsExceededException
public MaxFcnEvalsExceededException(string s, System.Exception exception)

Description
Maximum function evaluations exceeded.
Parameters

- The error message that explains the reason for the exception.

Exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

MaxFcnEvalsExceededException

```csharp
class MaxFcnEvalsExceededException
```

Description

Maximum function evaluations exceeded.

Parameters

- s – The error message that explains the reason for the exception.
- args – An Object array which specifies arguments used to format the message.

MaxFcnEvalsExceededException

```csharp
class MaxFcnEvalsExceededException
```

Description

Maximum function evaluations exceeded.

Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

MultipleSolutionsException Class

```csharp
class MultipleSolutionsException
```

The problem has multiple solutions producing essentially the same minimum.

Constructors

```csharp
MultipleSolutionsException
```

Exceptions

MultipleSolutionsException • 1835
Description
The problem has multiple solutions producing essentially the same minimum.

MultipleSolutionsException
public MultipleSolutionsException(string message)

Description
The problem has multiple solutions producing essentially the same minimum.

Parameter
message – The error message that explains the reason for the exception.

MultipleSolutionsException
public MultipleSolutionsException(string message, System.Exception exception)

Description
The problem has multiple solutions producing essentially the same minimum.

Parameters
message – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

MultipleSolutionsException

Description
The problem has multiple solutions producing essentially the same minimum.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

NoAcceptablePivotException Class

public class Imsl.Math.NoAcceptablePivotException : IMSLException : IISerializable

An acceptable pivot could not be found.
Constructors

NoAcceptablePivotException
public NoAcceptablePivotException()

Description
An acceptable pivot could not be found.

NoAcceptablePivotException
public NoAcceptablePivotException(string message)

Description
An acceptable pivot could not be found.

Parameter
message – The error message that explains the reason for the exception.

NoAcceptablePivotException
public NoAcceptablePivotException(string s, System.Exception exception)

Description
An acceptable pivot could not be found.

Parameters
s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NoAcceptablePivotException

Description
An acceptable pivot could not be found.

Parameters
info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

NoAcceptableStepsizeException Class

public class Imsl.Math.NoAcceptableStepsizeException : IMSLException : ISerializable

Exceptions
NoAcceptableStepsizeException • 1837
No acceptable stepsize in [SIGMA,SIGLA].

**Constructors**

NoAcceptableStepsizeException

public NoAcceptableStepsizeException(double sigma, double sigla)

**Description**

No acceptable stepsize in [SIGMA,SIGLA].

**Parameters**

sigma – A double containing the first messages argument SIGMA.

sigla – A double containing the second messages argument SIGLA.

NoAcceptableStepsizeException

public NoAcceptableStepsizeException()

**Description**

No acceptable stepsize in [SIGMA,SIGLA].

NoAcceptableStepsizeException

public NoAcceptableStepsizeException(string message)

**Description**

No acceptable stepsize in [SIGMA,SIGLA].

**Parameter**

message – The error message that explains the reason for the exception.

NoAcceptableStepsizeException

public NoAcceptableStepsizeException(string s, System.Exception exception)

**Description**

No acceptable stepsize in [SIGMA,SIGLA].

**Parameters**

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NoAcceptableStepsizeException

Description
No acceptable stepsize in [SIGMA,SIGLA].

Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

NoConvergenceException Class

public class Imsl.Math.NoConvergenceException : IMSLEException : ISerializable

The conjugate gradient method did not converge within the allowed maximum number of iterations.

Constructors

NoConvergenceException
public NoConvergenceException(int maxIterations)

Description
The conjugate gradient method did not converge within the allowed maximum number of iterations.

Parameter

- maxIterations – Maximum number of iterations exceeded.

NoConvergenceException
public NoConvergenceException(string message)

Description
The conjugate gradient method did not converge within the allowed maximum number of iterations.

Parameter

- message – The error message that explains the reason for the exception.

NoConvergenceException
public NoConvergenceException(string s, System.Exception exception)

Description
The conjugate gradient method did not converge within the allowed maximum number of iterations.

Exceptions

NoConvergenceException ● 1839
Parameters

- The error message that explains the reason for the exception.
- The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NoConvergenceException

Description
The conjugate gradient method did not converge within the allowed maximum number of iterations.

Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

NoLPSolutionException Class

public class Imsl.Math.NoLPSolutionException : IMSLException : ISerializable

No solution for the LP problem with \( h = 0 \) was found by DenseLP.

Constructors

NoLPSolutionException

public NoLPSolutionException(object[] args)

Description
No solution for the LP problem with \( h = 0 \) was found by DenseLP.

Parameter

- args – An Object array containing the input arguments to error string.

NoLPSolutionException

public NoLPSolutionException(string message)

Description
No solution for the LP problem with \( h = 0 \) was found by DenseLP.
Parameter

message – The error message that explains the reason for the exception.

NoLPSolutionException

public NoLPSolutionException(string message, System.Exception exception)

Description

No solution for the LP problem with \( h = 0 \) was found by DenseLP.

Parameters

message – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NoLPSolutionException


Description

No solution for the LP problem with \( h = 0 \) was found by DenseLP.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

NoProgressException Class

public class Imsl.Math.NoProgressException : IMSLException : ISerializable

The algorithm is not making any progress. Try a new initial guess.

Constructors

NoProgressException

public NoProgressException()

Description

The algorithm is not making any progress. Try a new initial guess.

NoProgressException

public NoProgressException(string message)
The algorithm is not making any progress. Try a new initial guess.

**Parameter**
- `message` – The error message that explains the reason for the exception.

---

**NoProgressException**

```csharp
public NoProgressException(string s, System.Exception exception)
```

**Description**
The algorithm is not making any progress. Try a new initial guess.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

---

**NoProgressException**

```csharp
NoProgressException(System.Runtime.Serialization.SerializationInfo info,
```

**Description**
The algorithm is not making any progress. Try a new initial guess.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**NotDefiniteAMatrixException Class**

```csharp
public class Imsl.Math.NotDefiniteAMatrixException : IMSLException : ISerializable
```

The input matrix A is indefinite, that is the matrix is not positive or negative definite.

---

**Constructors**

```csharp
NotDefiniteAMatrixException
public NotDefiniteAMatrixException()
```

---
Description
The input matrix A is indefinite, that is the matrix is not positive or negative definite.

**NotDefiniteAMatrixException**

```csharp
public NotDefiniteAMatrixException(string message)
```

**Description**
The input matrix A is indefinite, that is the matrix is not positive or negative definite.

**Parameter**
- `message` – The error message that explains the reason for the exception.

```csharp
public NotDefiniteAMatrixException(string s, System.Exception exception)
```

**Description**
The input matrix A is indefinite, that is the matrix is not positive or negative definite.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```csharp
```

**Description**
The input matrix A is indefinite, that is the matrix is not positive or negative definite.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**NotDefiniteJacobiPreconditionerException Class**

```csharp
public class Imsl.Math.NotDefiniteJacobiPreconditionerException : IMSLException : ISerializable
```

The Jacobi preconditioner is not strictly positive or negative definite.

**Exceptions**

- `NotDefiniteJacobiPreconditionerException` • 1843
Constructors

**NotDefiniteJacobiPreconditionerException**

public NotDefiniteJacobiPreconditionerException()

**Description**
The Jacobi preconditioner is not strictly positive or negative definite.

**NotDefiniteJacobiPreconditionerException**

public NotDefiniteJacobiPreconditionerException(string message)

**Description**
The Jacobi preconditioner is not strictly positive or negative definite.

**Parameter**
message – The error message that explains the reason for the exception.

**NotDefiniteJacobiPreconditionerException**

public NotDefiniteJacobiPreconditionerException(string s, System.Exception exception)

**Description**
The Jacobi preconditioner is not strictly positive or negative definite.

**Parameters**
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**NotDefiniteJacobiPreconditionerException**


**Description**
The Jacobi preconditioner is not strictly positive or negative definite.

**Parameters**
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.
**NotDefinitePreconditionMatrixException Class**

```csharp
public class Imsl.Math.NotDefinitePreconditionMatrixException : IMSLEException
    : ISerializable

The Precondition matrix is indefinite.
```

**Constructors**

**NotDefinitePreconditionMatrixException**

```csharp
public NotDefinitePreconditionMatrixException()
```

**Description**

The Precondition matrix is indefinite.

**NotDefinitePreconditionMatrixException**

```csharp
public NotDefinitePreconditionMatrixException(string message)
```

**Description**

The Precondition matrix is indefinite.

**Parameter**

*message* – The error message that explains the reason for the exception.

**NotDefinitePreconditionMatrixException**

```csharp
public NotDefinitePreconditionMatrixException(string s, System.Exception exception)
```

**Description**

The Precondition matrix is indefinite.

**Parameters**

*s* – The error message that explains the reason for the exception.

*exception* – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**NotDefinitePreconditionMatrixException**

```csharp
```

**Description**

The Precondition matrix is indefinite.

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotDefinitePreconditionMatrixException • 1845</td>
<td>Indefinite Precondition matrix exception</td>
</tr>
</tbody>
</table>
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

NotSPDEexception Class

public class Imsl.Math.NotSPDEexception : IMSLException : ISerializable

The matrix is not symmetric, positive definite.

Constructors

NotSPDEexception

public NotSPDEexception()

Description

The matrix is not symmetric, positive definite.

NotSPDEexception

public NotSPDEexception(string message)

Description

The matrix is not symmetric, positive definite.

Parameter

message – The error message that explains the reason for the exception.

NotSPDEexception

public NotSPDEexception(string s, System.Exception exception)

Description

The matrix is not symmetric, positive definite.

Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NotSPDEexception

**Description**
The matrix is not symmetric, positive definite.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**NumericDifficultyException Class**

```csharp
public class Imsl.Math.NumericDifficultyException : IMSLEmotion : ISerializable

Numerical difficulty occurred.
```

**Constructors**

**NumericDifficultyException**
```csharp
public NumericDifficultyException()
```

**Description**
Numerical difficulty occurred.

**NumericDifficultyException**
```csharp
public NumericDifficultyException(string message)
```

**Description**
Numerical difficulty occurred.

**Parameter**
- `message` – The error message that explains the reason for the exception.

**NumericDifficultyException**
```csharp
public NumericDifficultyException(string s, System.Exception exception)
```

**Description**
Numerical difficulty occurred.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**Exceptions**

```
NumericDifficultyException  ●  1847
```
**NumericDifficultyException**


**Description**
Numerical difficulty occurred.

**Parameters**
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**ObjectiveEvaluationException Class**

```csharp
public class Imsl.Math.ObjectiveEvaluationException : IMSLException : ISerializable
Objective evaluation returns an error with current point.
```

**Constructors**

**ObjectiveEvaluationException**

```csharp
public ObjectiveEvaluationException()
```

**Description**
Objective evaluation returns an error with current point.

**ObjectiveEvaluationException**

```csharp
public ObjectiveEvaluationException(string message)
```

**Description**
Objective evaluation returns an error with current point.

**Parameter**
- message – The error message that explains the reason for the exception.

**ObjectiveEvaluationException**

```csharp
public ObjectiveEvaluationException(string s, System.Exception exception)
```

**Description**
Objective evaluation returns an error with current point.
Parameters
- The error message that explains the reason for the exception.
- The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**ObjectiveEvaluationException**

Description
Objective evaluation returns an error with current point.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**PenaltyFunctionPointInfeasibleException Class**

public class Imsl.Math.PenaltyFunctionPointInfeasibleException : IMSLException : ISerializable

Penalty function point infeasible.

**Constructors**

---

**PenaltyFunctionPointInfeasibleException**
public PenaltyFunctionPointInfeasibleException()

Description
Penalty function point infeasible.

**PenaltyFunctionPointInfeasibleException**
public PenaltyFunctionPointInfeasibleException(string message)

Description
Penalty function point infeasible.
Parameter

message – The error message that explains the reason for the exception.

PenaltyFunctionPointInfeasibleException

public PenaltyFunctionPointInfeasibleException(string s, System.Exception exception)

Description

Penalty function point infeasible.

Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

PenaltyFunctionPointInfeasibleException


Description

Penalty function point infeasible.

Parameters

info – The object that holds the serialized object data.

collection – The contextual information about the source or destination.

ProblemInfeasibleException Class

public class Imsl.Math.ProblemInfeasibleException : IMSLException : ISerializable

The problem is not feasible. The constraints are inconsistent.

Constructors

ProblemInfeasibleException

public ProblemInfeasibleException()
Description
The problem is not feasible. The constraints are inconsistent.

**ProblemInfeasibleException**

```csharp
public ProblemInfeasibleException(string message)
```

Description
The problem is not feasible. The constraints are inconsistent.

Parameter

`message` – The error message that explains the reason for the exception.

```csharp
public ProblemInfeasibleException(string s, System.Exception exception)
```

Description
The problem is not feasible. The constraints are inconsistent.

Parameters

`s` – The error message that explains the reason for the exception.

`exception` – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```csharp
```

Description
The problem is not feasible. The constraints are inconsistent.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

---

**ProblemUnboundedException Class**

```csharp
public class Imsl.Math.ProblemUnboundedException : IMSLException : ISerializable
```

The problem is unbounded.

---

**Exceptions**

| ProblemUnboundedException | 1851 |
Constructors

ProblemUnboundedException
public ProblemUnboundedException()

Description
The problem is unbounded.

ProblemUnboundedException
public ProblemUnboundedException(string message)

Description
The problem is unbounded.

Parameter
message – The error message that explains the reason for the exception.

ProblemUnboundedException
public ProblemUnboundedException(string s, System.Exception exception)

Description
The problem is unbounded.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

ProblemUnboundedException
ProblemUnboundedException(System.Runtime.Serialization.SerializationInfo info,

Description
The problem is unbounded.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

ProblemVacuousException Class

public class Imsl.Math.ProblemVacuousException : IMSLException : ISerializable

1852 • ProblemVacuousException IML C# Numerical Library
The problem is vacuous.

**Constructors**

```csharp
ProblemVacuousException
public ProblemVacuousException()
```

**Description**
The problem is vacuous.

```csharp
ProblemVacuousException
public ProblemVacuousException(string message)
```

**Description**
The problem is vacuous.

**Parameter**

- **message** – The error message that explains the reason for the exception.

```csharp
ProblemVacuousException
public ProblemVacuousException(string s, System.Exception exception)
```

**Description**
The problem is vacuous.

**Parameters**

- **s** – The error message that explains the reason for the exception.
- **exception** – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```csharp
ProblemVacuousException
```

**Description**
The problem is vacuous.

**Parameters**

- **info** – The object that holds the serialized object data.
- **context** – The contextual information about the source or destination.

**Exceptions**

- ProblemVacuousException • 1853
QPInfeasibleException Class

public class Imsl.Math.QPInfeasibleException : IMSLException : ISerializable

QP problem seemingly infeasible.

Constructors

QPInfeasibleException
public QPInfeasibleException()

Description
QP problem seemingly infeasible.

QPInfeasibleException
public QPInfeasibleException(string message)

Description
QP problem seemingly infeasible.

Parameter
message – The error message that explains the reason for the exception.

QPInfeasibleException
public QPInfeasibleException(string s, System.Exception exception)

Description
QP problem seemingly infeasible.

Parameters
s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

QPInfeasibleException

Description
QP problem seemingly infeasible.
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

QPProblemUnboundedException Class

```csharp
public class Imsl.Math.QPProblemUnboundedException : IMSLException : ISerializable

The object value for the problem is unbounded.
```

Constructors

- **QPProblemUnboundedException**
  ```csharp
  public QPProblemUnboundedException()
  ```
  **Description**
  The object value for the problem is unbounded.

- **QPProblemUnboundedException**
  ```csharp
  public QPProblemUnboundedException(string message)
  ```
  **Description**
  The object value for the problem is unbounded.
  **Parameter**
  - `message` – The error message that explains the reason for the exception.

- **QPProblemUnboundedException**
  ```csharp
  public QPProblemUnboundedException(string message, System.Exception exception)
  ```
  **Description**
  The object value for the problem is unbounded.
  **Parameters**
  - `message` – The error message that explains the reason for the exception.
  - `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

Exceptions

- QPProblemUnboundedException • 1855
QPProblemUnboundedException


Description
The object value for the problem is unbounded.

Parameters

  info – The object that holds the serialized object data.
  context – The contextual information about the source or destination.

SingularException Class

public class Imsl.Math.SingularException : IMSLException : ISerializable

Problem is singular.

Constructors

SingularException

public SingularException()

Description
Problem is singular.

SingularException

public SingularException(string message)

Description
Problem is singular.

Parameter

  message – The error message that explains the reason for the exception.

SingularException

public SingularException(string s, System.Exception exception)

Description
Problem is singular.
Parameters

- The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

SingularException

SingularException(System.Runtime.Serialization.SerializationInfo info,

Description
Problem is singular.

Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

SingularMatrixException Class

public class Imsl.Math.SingularMatrixException : IMSLException : ISerializable

The matrix is singular.

Constructors

SingularMatrixException

custom
public SingularMatrixException()

Description
The matrix is singular.

SingularMatrixException

custom
public SingularMatrixException(string message)

Description
The matrix is singular.

Exceptions

SingularMatrixException • 1857
Parameter

- message – The error message that explains the reason for the exception.

**SingularMatrixException**

```csharp
public SingularMatrixException(string s, System.Exception exception)
```

**Description**

The matrix is singular.

**Parameters**

- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**SingularMatrixException**

```csharp
```

**Description**

The matrix is singular.

**Parameters**

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

**SingularPreconditionMatrixException Class**

```csharp
public class Imsl.Math.SingularPreconditionMatrixException : IMSLException : ISerializable
```

The Precondition matrix is singular.

**Constructors**

```csharp
public SingularPreconditionMatrixException()
```

---

1858 • **SingularPreconditionMatrixException** • IMSL C# Numerical Library
**SingularPreconditionMatrixException**

```csharp
public SingularPreconditionMatrixException(string message)
```

**Description**
The Precondition matrix is singular.

**Parameter**
- `message` – The error message that explains the reason for the exception.

```csharp
public SingularPreconditionMatrixException(string s, System.Exception exception)
```

**Description**
The Precondition matrix is singular.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```csharp
```

**Description**
The Precondition matrix is singular.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**SolutionNotFoundException Class**

```csharp
public class Imsl.Math.SolutionNotFoundException : IMSLException : ISerializable
```

A solution was not found. Try using DenseLP.
Constructors

**SolutionNotFoundException**

```csharp
public SolutionNotFoundException()
```

**Description**
A solution was not found. Try using DenseLP.

**SolutionNotFoundException**

```csharp
public SolutionNotFoundException(string message)
```

**Description**
A solution was not found. Try using DenseLP.

**Parameter**

- `message` – The error message that explains the reason for the exception.

**SolutionNotFoundException**

```csharp
public SolutionNotFoundException(string message, System.Exception exception)
```

**Description**
A solution was not found. Try using DenseLP.

**Parameters**

- `message` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**SolutionNotFoundException**

```csharp
```

**Description**
A solution was not found. Try using DenseLP.

**Parameters**

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**SomeConstraintsDiscardedException Class**

```csharp
public class Imsl.Math.SomeConstraintsDiscardedException : IMSLException, ISerializable
```

**SomeConstraintsDiscardedException**

```csharp
1860 • SomeConstraintsDiscardedException
```
Some constraints were discarded because they were too linearly dependent on other active constraints.

**Constructors**

```csharp
public SomeConstraintsDiscardedException()
Description
Some constraints were discarded because they were too linearly dependent on other active constraints.
```

```csharp
public SomeConstraintsDiscardedException(string message)
Description
Some constraints were discarded because they were too linearly dependent on other active constraints.
Parameter
message – The error message that explains the reason for the exception.
```

```csharp
public SomeConstraintsDiscardedException(string message, System.Exception exception)
Description
Some constraints were discarded because they were too linearly dependent on other active constraints.
Parameters
message – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.
```

```csharp
Description
Some constraints were discarded because they were too linearly dependent on other active constraints.
Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.
```

**Exceptions**

SomeConstraintsDiscardedException • 1861
TcurrentTstopInconsistentException Class

public class Imsl.Math.TcurrentTstopInconsistentException : IMSLException : ISerializable

The end value for the integration in time, tout, is not consistent with the current time value, t.

Constructors

TcurrentTstopInconsistentException

public TcurrentTstopInconsistentException(double time, double stepsize, double tstop)

Description
The end value for the integration in time, tout, is not consistent with the current time value, t.

Parameters
- time – A double scalar, the current time value.
- stepsize – A double scalar, the stepsize used in the integration.
- tstop – A double scalar, the integration endpoint.

TcurrentTstopInconsistentException

public TcurrentTstopInconsistentException()

Description
The end value for the integration in time, tout, is not consistent with the current time value, t.

TcurrentTstopInconsistentException

public TcurrentTstopInconsistentException(string s, object[] args)

Description
The end value for the integration in time, tout, is not consistent with the current time value, t.

Parameters
- s – The error message that explains the reason for the exception.
- args – Arguments for the error message.

TcurrentTstopInconsistentException

public TcurrentTstopInconsistentException(string message)

Description
The end value for the integration in time, tout, is not consistent with the current time value, t.
Parameter

message – The error message that explains the reason for the exception.

TcurrentTstopInconsistentException
public TcurrentTstopInconsistentException(string s, System.Exception exception)

Description
The end value for the integration in time, tout, is not consistent with the current time value, t.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

TcurrentTstopInconsistentException

Description
The end value for the integration in time, tout, is not consistent with the current time value, t.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

TEqualsToutException Class

public class Imsl.Math.TEqualsToutException : IMSLException : ISerializable
The current integration point in time and the end point are equal.

Constructors

TEqualsToutException
public TEqualsToutException(double time)

Description
The current integration point in time and the end point are equal.

Exceptions

TEqualsToutException • 1863
Parameter
    time – A double scalar, the current time point which is equal to the end point.

**TEqualsToutException**

```csharp
public TEqualsToutException()
```

**Description**
The current integration point in time and the end point are equal.

**TEqualsToutException**

```csharp
public TEqualsToutException(string s, object[] args)
```

**Description**
The current integration point in time and the end point are equal.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `args` – Arguments for the error message.

**TEqualsToutException**

```csharp
public TEqualsToutException(string message)
```

**Description**
The current integration point in time and the end point are equal.

**Parameter**
- `message` – The error message that explains the reason for the exception.

**TEqualsToutException**

```csharp
public TEqualsToutException(string s, System.Exception exception)
```

**Description**
The current integration point in time and the end point are equal.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**TEqualsToutException**

```csharp
```

**Description**
The current integration point in time and the end point are equal.

---

1864 • TEqualsToutException  IMSL C# Numerical Library
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

TerminationCriteriaNotSatisfiedException Class

```java
public class Imsl.Math.TerminationCriteriaNotSatisfiedException : IMSLException : ISerializable
Termination criteria are not satisfied.
```

---

Constructors

**TerminationCriteriaNotSatisfiedException**

```java
public TerminationCriteriaNotSatisfiedException(int numsm)
```

**Description**
Termination criteria are not satisfied.

**Parameter**
- numsm – An int containing the criteria value.

```java
public TerminationCriteriaNotSatisfiedException()
```

**Description**
Termination criteria are not satisfied.

```java
public TerminationCriteriaNotSatisfiedException(string message)
```

**Description**
Termination criteria are not satisfied.

**Parameter**
- message – The error message that explains the reason for the exception.

```java
public TerminationCriteriaNotSatisfiedException(string s, System.Exception exception)
```

---

Exceptions

TerminationCriteriaNotSatisfiedException • 1865
Termination criteria are not satisfied.

Parameters
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**TerminationCriteriaNotSatisfiedException**

Description
Termination criteria are not satisfied.

Parameters
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**TimeIntervalTooSmallException Class**

```csharp
public class Imsl.Math.TimeIntervalTooSmallException : IMSLException : ISerializable

Distance between starting time point and end point for the integration is too small.
```

**Constructors**

**TimeIntervalTooSmallException**

```csharp
public TimeIntervalTooSmallException(double tend, double tstart)
```

Description
Distance between starting time point and end point for the integration is too small.

Parameters
- `tend` – A double scalar, the end point of the integration.
- `tstart` – A double scalar, the starting point for the integration.

**TimeIntervalTooSmallException**

```csharp
public TimeIntervalTooSmallException()
```
Distance between starting time point and end point for the integration is too small.

TimeIntervalTooSmallException

public TimeIntervalTooSmallException(string s, object[] args)

Description
Distance between starting time point and end point for the integration is too small.

Parameters
s – The error message that explains the reason for the exception.
args – Arguments for the error message.

TimeIntervalTooSmallException

public TimeIntervalTooSmallException(string message)

Description
Distance between starting time point and end point for the integration is too small.

Parameter
message – The error message that explains the reason for the exception.

TimeIntervalTooSmallException

public TimeIntervalTooSmallException(string s, System.Exception exception)

Description
Distance between starting time point and end point for the integration is too small.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

TimeIntervalTooSmallException


Description
Distance between starting time point and end point for the integration is too small.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

Exceptions TimeIntervalTooSmallException • 1867
ToleranceTooSmallException Class

public class Imsl.Math.ToleranceTooSmallException : IMSLException : ISerializable
Tolerance is too small.

Constructors

ToleranceTooSmallException
public ToleranceTooSmallException()

Description
Tolerance is too small.

ToleranceTooSmallException
public ToleranceTooSmallException(double tol)

Description
Tolerance is too small.

Parameter
tol – A double containing the tolerance value.

ToleranceTooSmallException
public ToleranceTooSmallException(string message)

Description
Tolerance is too small.

Parameter
message – The error message that explains the reason for the exception.

ToleranceTooSmallException
public ToleranceTooSmallException(string s, System.Exception exception)

Description
Tolerance is too small.

Parameters
s – The error message that explains the reason for the exception.
expection – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.
ToleranceTooSmallException

```java
public ToleranceTooSmallException(string s, object[] args)
```

**Description**
Tolerance is too small.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `args` – An `Object` array which specifies arguments used to format the message.

ToleranceTooSmallException

```java
```

**Description**
Tolerance is too small.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

**TooManyIterationsException Class**

```java
public class Imsl.Math.TooManyIterationsException : IMSLException : ISerializable
```

Maximum number of iterations exceeded.

**Constructors**

**TooManyIterationsException**

```java
public TooManyIterationsException(int maximumNumberOfIterations)
```

**Description**
Maximum number of iterations exceeded.

**Parameter**
- `maximumNumberOfIterations` – Maximum number of iterations allowed.

**TooManyIterationsException**

```java
public TooManyIterationsException()
```

**Exceptions**

TooManyIterationsException • 1869
**Description**

Maximum number of iterations exceeded.

**TooManyIterationsException**

public TooManyIterationsException(string message)

**Description**

Maximum number of iterations exceeded.

**Parameter**

message – The error message that explains the reason for the exception.

**TooManyIterationsException**

public TooManyIterationsException(string s, object[] args)

**Description**

Maximum number of iterations exceeded.

**Parameters**

s – The error message that explains the reason for the exception.

args – Arguments for the error message.

**TooManyIterationsException**

public TooManyIterationsException(string s, System.Exception exception)

**Description**

Maximum number of iterations exceeded.

**Parameters**

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**TooManyIterationsException**


**Description**

Maximum number of iterations exceeded.

**Parameters**

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.
TooManyStepsException Class

public class Imsl.Math.TooManyStepsException : IMSLException : ISerializable
Too many steps were taken between two consecutive time steps.

Constructors

TooManyStepsException
public TooManyStepsException(double t0, int steps, double t1)

Description
Too many steps were taken between two consecutive time steps.

Parameters
  t0 – The current time value.
  steps – The maximum number of iterations allowed.
  t1 – The end point of the internal integration in time.

TooManyStepsException
public TooManyStepsException()  

Description
Too many steps were taken between two consecutive time steps.

TooManyStepsException
public TooManyStepsException(string message)

Description
Too many steps were taken between two consecutive time steps.

Parameter
  message – The error message that explains the reason for the exception.

TooManyStepsException
public TooManyStepsException(string s, System.Exception exception)

Description
Too many steps were taken between two consecutive time steps.

Exceptions

TooManyStepsException • 1871
### Parameters
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

#### TooManyStepsException
**TooManyStepsException**

**Description**
Too many steps were taken between two consecutive time steps.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

### TooMuchTimeException Class

```
public class Imsl.Math.TooMuchTimeException : IMSLException : ISerializable
```

Maximum time allowed for solve is exceeded.

### Constructors

#### TooMuchTimeException
**TooMuchTimeException**
public TooMuchTimeException(long maximumTime)

**Description**
Maximum time allowed for solve is exceeded.

**Parameter**
- `maximumTime` – The maximum time allowed for the solve step.

**TooMuchTimeException**
public TooMuchTimeException()

**Description**
Maximum time allowed for solve is exceeded.

**TooMuchTimeException**
public TooMuchTimeException(string message)
Description
Maximum time allowed for solve is exceeded.

Parameter
- message – The error message that explains the reason for the exception.

**TooMuchTimeException**
public TooMuchTimeException(string s, System.Exception exception)

Description
Maximum time allowed for solve is exceeded.

Parameters
- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**TooMuchTimeException**

Description
Maximum time allowed for solve is exceeded.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**UnboundedBelowException Class**

```java
public class Imsl.Math.UnboundedBelowException : IMSLException : ISerializable
```

Five consecutive steps of the maximum allowable stepsize have been taken, either the function is unbounded below, or has a finite asymptote in some direction or the maximum allowable step size is too small.

**Constructors**

**UnboundedBelowException**
public UnboundedBelowException()
Description
Five consecutive steps of the maximum allowable stepsize have been taken, either the function is
unbounded below, or has a finite asymptote in some direction or the maximum allowable step size is too
small.

UnboundedBelowException
public UnboundedBelowException(string message)

Description
Five consecutive steps of the maximum allowable stepsize have been taken, either the function is
unbounded below, or has a finite asymptote in some direction or the maximum allowable step size is too
small.

Parameter
message – The error message that explains the reason for the exception.

UnboundedBelowException
public UnboundedBelowException(string s, System.Exception exception)

Description
Five consecutive steps of the maximum allowable stepsize have been taken, either the function is
unbounded below, or has a finite asymptote in some direction or the maximum allowable step size is too
small.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException
parameter is not a null reference, the current exception is raised in a catch block that handles the
inner exception.

UnboundedBelowException
UnboundedBelowException(System.Runtime.Serialization.SerializationInfo info,

Description
Five consecutive steps of the maximum allowable stepsize have been taken, either the function is
unbounded below, or has a finite asymptote in some direction or the maximum allowable step size is too
small.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.
VarBoundsInconsistentException Class

public class Imsl.Math.VarBoundsInconsistentException : IMSLException : ISerializable

The equality constraints and the bounds on the variables are found to be inconsistent.

Constructors

VarBoundsInconsistentException
public VarBoundsInconsistentException()

Description
The equality constraints and the bounds on the variables are found to be inconsistent.

VarBoundsInconsistentException
public VarBoundsInconsistentException(string message)

Description
The equality constraints and the bounds on the variables are found to be inconsistent.

Parameter
message – The error message that explains the reason for the exception.

VarBoundsInconsistentException
public VarBoundsInconsistentException(string s, System.Exception exception)

Description
The equality constraints and the bounds on the variables are found to be inconsistent.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

VarBoundsInconsistentException

Description
The equality constraints and the bounds on the variables are found to be inconsistent.

Exceptions
VarBoundsInconsistentException • 1875
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

**WorkingSetSingularException Class**

```csharp
public class Imsl.Math.WorkingSetSingularException : IMSLException : ISerializable
{
    Working set is singular in dual extended QP.
}
```

**Constructors**

**WorkingSetSingularException**

```csharp
public WorkingSetSingularException()
{
    Description
    Working set is singular in dual extended QP.
}
```

**WorkingSetSingularException**

```csharp
public WorkingSetSingularException(string message)
{
    Description
    Working set is singular in dual extended QP.
    Parameter
    message – The error message that explains the reason for the exception.
}
```

**WorkingSetSingularException**

```csharp
public WorkingSetSingularException(string s, System.Exception exception)
{
    Description
    Working set is singular in dual extended QP.
    Parameters
    s – The error message that explains the reason for the exception.
    exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.
}
```
WorkingSetSingularException


Description
Working set is singular in dual extended QP.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

AllDeletedException Class

public class Imsl.Stat.AllDeletedException : IMSLException : ISerializable
There are no observations.

Constructors

AllDeletedException
public AllDeletedException()

Description
There are no observations.

AllDeletedException
public AllDeletedException(string message)

Description
There are no observations.

Parameter
- message – The error message that explains the reason for the exception.

AllDeletedException
public AllDeletedException(string s, System.Exception exception)

Description
There are no observations.

Exceptions

AllDeletedException • 1877
Parameters

- **s** – The error message that explains the reason for the exception.
- **exception** – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

---

**AllDeletedException**


Description

There are no observations.

Parameters

- **info** – The object that holds the serialized object data.
- **context** – The contextual information about the source or destination.

---

**AllMissingException Class**

public class Imsl.Stat.AllMissingException : IMSLException : ISerializable

There are no observations.

---

**Constructors**

---

**AllMissingException**

public AllMissingException()

Description

There are no observations.

---

**AllMissingException**

public AllMissingException(string message)

Description

There are no observations.

Parameter

- **message** – The error message that explains the reason for the exception.

---

**AllMissingException**

public AllMissingException(string s, System.Exception exception)
Description
There are no observations.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

AllMissingException

Description
There are no observations.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

AltSeriesAccuracyLossException Class

public class Imsl.Stat.AltSeriesAccuracyLossException : IMSLEException : ISerializable
The magnitude of alternating series sum is too small relative to the sum of positive terms to permit a reliable accuracy.

Constructors

AltSeriesAccuracyLossException
public AltSeriesAccuracyLossException()

Description
The magnitude of alternating series sum is too small relative to the sum of positive terms to permit a reliable accuracy.

AltSeriesAccuracyLossException
public AltSeriesAccuracyLossException(string message)

Exceptions
AltSeriesAccuracyLossException • 1879
Description
The magnitude of alternating series sum is too small relative to the sum of positive terms to permit a reliable accuracy.

Parameter
- message – The error message that explains the reason for the exception.

AltSeriesAccuracyLossException
public AltSeriesAccuracyLossException(string s, System.Exception exception)

Description
The magnitude of alternating series sum is too small relative to the sum of positive terms to permit a reliable accuracy.

Parameters
- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

AltSeriesAccuracyLossException

Description
The magnitude of alternating series sum is too small relative to the sum of positive terms to permit a reliable accuracy.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

BadVarianceException Class

public class Imsl.Stat.BadVarianceException : IMSLException : ISerializable
The input variance is not in the allowed range.

 Constructors

BadVarianceException
public BadVarianceException(int i, double cov, double uniq)
Description
The input variance is not in the allowed range.

Parameters
- i – A int specifying the index of variable uniq, causing the error.
- cov – A double specifying the value of cov[i,i].
- uniq – A double specifying the input variance.

BadVarianceException
public BadVarianceException()

Description
Maximum number of iterations exceeded.

BadVarianceException
public BadVarianceException(string message)

Description
Maximum number of iterations exceeded.

Parameter
- message – The error message that explains the reason for the exception.

BadVarianceException
public BadVarianceException(string s, System.Exception exception)

Description
Maximum number of iterations exceeded.

Parameters
- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

BadVarianceException

Description
Maximum number of iterations exceeded.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

Exceptions
BadVarianceException • 1881
ClassificationVariableException Class

```csharp
public class Imsl.Stat.ClassificationVariableException : IMSLException : ISerializable
The ClassificationVariable vector has not been initialized.
```

Constructors

**ClassificationVariableException**
```csharp
public ClassificationVariableException()
```
**Description**
The ClassificationVariable vector has not been initialized.

**ClassificationVariableException**
```csharp
public ClassificationVariableException(string message)
```
**Description**
The ClassificationVariable vector has not been initialized.
**Parameter**
- `message` – The error message that explains the reason for the exception.

**ClassificationVariableException**
```csharp
public ClassificationVariableException(string s, System.Exception exception)
```
**Description**
The ClassificationVariable vector has not been initialized.
**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**ClassificationVariableException**
```csharp
```
**Description**
The ClassificationVariable vector has not been initialized.
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

ClassificationVariableLimitException Class

public class Imsl.Stat.ClassificationVariableLimitException : IMSLException : ISerializable

The Classification Variable limit set by the user through `setUpperBound` has been exceeded.

Constructors

ClassificationVariableLimitException
public ClassificationVariableLimitException(int maxcl)

Description
The Classification Variable limit set by the user through `setUpperBound` has been exceeded.

Parameter
- `maxcl`– An int which specifies the upper bound.

ClassificationVariableLimitException
public ClassificationVariableLimitException()

Description
The Classification Variable limit set by the user through `setUpperBound` has been exceeded.

ClassificationVariableLimitException
public ClassificationVariableLimitException(string message)

Description
The Classification Variable limit set by the user through `setUpperBound` has been exceeded.

Parameter
- `message` – The error message that explains the reason for the exception.

ClassificationVariableLimitException
public ClassificationVariableLimitException(string s, System.Exception exception)

Exceptions ClassificationVariableLimitException • 1883
Description
The Classification Variable limit set by the user through `setUpperBound` has been exceeded.

Parameters
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

---

ClassificationVariableLimitException

```csharp
public ClassificationVariableLimitException(string s, object[] args)
```

Description
The Classification Variable limit set by the user through `setUpperBound` has been exceeded.

Parameters
- `s` – The error message that explains the reason for the exception.
- `args` – An `Object` array which specifies arguments used to format the message.

ClassificationVariableLimitException

```csharp
```

Description
The Classification Variable limit set by the user through `setUpperBound` has been exceeded.

Parameters
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

ClassificationVariableValueException Class

```csharp
public class Imsl.Stat.ClassificationVariableValueException : IMSLException : ISerializable
```

The number of distinct values for each Classification Variable must be greater than 1.

---

Constructors

ClassificationVariableValueException

```csharp
public ClassificationVariableValueException(int index, int val)
```
Description
The number of distinct values for each Classification Variable must be greater than 1.

Parameters
- index – An int which specifies the index of a classification variable.
- val – An int which specifies the number of distinct values that can be taken by this classification variable.

ClassificationVariableValueException
public ClassificationVariableValueException()

Description
The number of distinct values for each Classification Variable must be greater than 1.

ClassificationVariableValueException
public ClassificationVariableValueException(string message)

Description
The number of distinct values for each Classification Variable must be greater than 1.

Parameter
message – The error message that explains the reason for the exception.

ClassificationVariableValueException
public ClassificationVariableValueException(string s, System.Exception exception)

Description
The number of distinct values for each Classification Variable must be greater than 1.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

ClassificationVariableValueException

Description
The number of distinct values for each Classification Variable must be greater than 1.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

Exceptions
ClassificationVariableValueException • 1885
ClusterNoPointsException Class

public class Imsl.Stat.ClusterNoPointsException : IMSLException : ISerializable
There is a cluster with no points.

Constructors

ClusterNoPointsException
public ClusterNoPointsException()

Description
There is a cluster with no points.

ClusterNoPointsException
public ClusterNoPointsException(int clusterNumber)

Description
There is a cluster with no points.

Parameter
clusterNumber – Number of the cluster with no points.

ClusterNoPointsException
public ClusterNoPointsException(string message)

Description
There is a cluster with no points.

Parameter
message – The error message that explains the reason for the exception.

ClusterNoPointsException
public ClusterNoPointsException(string s, System.Exception exception)

Description
There is a cluster with no points.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.
ClusterNoPointsException

Description
There is a cluster with no points.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

ConstrInconsistentException Class

public class Imsl.Stat.ConstrInconsistentException : IMSLException : ISerializable
The equality constraints are inconsistent.

Constructors

ConstrInconsistentException
public ConstrInconsistentException()

Description
The equality constraints are inconsistent.

ConstrInconsistentException
public ConstrInconsistentException(string message)

Description
The equality constraints are inconsistent.

Parameter
- message – The error message that explains the reason for the exception.

ConstrInconsistentException
public ConstrInconsistentException(string s, System.Exception exception)

Description
The equality constraints are inconsistent.

Exceptions
ConstrInconsistentException • 1887
**Parameters**

s – The error message that explains the reason for the exception.

e – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

---

**ConstrInconsistentException**


**Description**
The equality constraints are inconsistent.

**Parameters**

info – The object that holds the serialized object data.

c – The contextual information about the source or destination.

---

**CovarianceSingularException Class**

```csharp
public class Imsl.Stat.CovarianceSingularException : IMSLException : ISerializable
```

The variance-Covariance matrix is singular.

---

**Constructors**

**CovarianceSingularException**

```csharp
public CovarianceSingularException()
```

**Description**
The variance-Covariance matrix is singular.

**CovarianceSingularException**

```csharp
public CovarianceSingularException(int l)
```

**Description**
The variance-Covariance matrix is singular.
Parameter

l – An int which specifies the population number.

CovarianceSingularException
def public CovarianceSingularException(string message)

Description

The variance-Covariance matrix is singular.

Parameter

message – The error message that explains the reason for the exception.

CovarianceSingularException
def public CovarianceSingularException(string s, System.Exception exception)

Description

The variance-Covariance matrix is singular.

Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException
parameter is not a null reference, the current exception is raised in a catch block that handles the
inner exception.

CovarianceSingularException
def CovarianceSingularException(System.Runtime.Serialization.SerializationInfo info,

Description

The variance-Covariance matrix is singular.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

CovarianceSingular1Exception Class

public class Imsl.Stat.CovarianceSingular1Exception :
CovarianceSingularException : ISerializable

The variance-Covariance matrix is singular.

Exceptions

CovarianceSingular1Exception • 1889
Constructor

**CovarianceSingular1Exception**

public CovarianceSingular1Exception(int l)

**Description**
The variance-Covariance matrix is singular.

**Parameter**

1 – An int which specifies the population number.

---

**CovarianceSingular2Exception Class**

public class Imsl.Stat.CovarianceSingular2Exception : CovarianceSingularException : ISerializable

The variance-Covariance matrix is singular.

Constructor

**CovarianceSingular2Exception**

public CovarianceSingular2Exception()

**Description**
The variance-Covariance matrix is singular.

---

**CyclingIsOccurringException Class**

public class Imsl.Stat.CyclingIsOccurringException : IMSLException : ISerializable

Cycling is occurring.
Constructors

**CyclingIsOccurringException**
public CyclingIsOccurringException(int nStep)

**Description**
Cycling is occurring.

**Parameter**
- nStep – An int which specifies the number of steps taken.

**CyclingIsOccurringException**
public CyclingIsOccurringException()

**Description**
Cycling is occurring.

**CyclingIsOccurringException**
public CyclingIsOccurringException(string message)

**Description**
Cycling is occurring.

**Parameter**
- message – The error message that explains the reason for the exception.

**CyclingIsOccurringException**
public CyclingIsOccurringException(string s, System.Exception exception)

**Description**
Cycling is occurring.

**Parameters**
- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**CyclingIsOccurringException**

**Description**
Cycling is occurring.

**Parameters**
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

Exceptions

| CyclingIsOccurringException | 1891 |
DeleteObservationsException Class

```java
public class Imsl.Stat.DeleteObservationsException : IMSLException : ISerializable
```

The number of observations to be deleted (set by `setObservationMax`) has grown too large.

Constructors

```java
DeleteObservationsException
public DeleteObservationsException(int nmax)
```

Description

The number of observations to be deleted (set with `ObservationMax`) has grown too large.

Parameter

- `nmax` – An `int` which specifies the maximum number of observations that can be handled in the linear programming.

```java
DeleteObservationsException
public DeleteObservationsException()
```

Description

The number of observations to be deleted (set by `setObservationMax`) has grown too large.

```java
DeleteObservationsException
public DeleteObservationsException(string message)
```

Description

The number of observations to be deleted (set by `setObservationMax`) has grown too large.

Parameter

- `message` – The error message that explains the reason for the exception.

```java
DeleteObservationsException
public DeleteObservationsException(string s, System.Exception exception)
```

Description

The number of observations to be deleted (set by `setObservationMax`) has grown too large.
**Parameters**

- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

---

**DeleteObservationsException**


**Description**
The number of observations to be deleted (set by `setObservationMax`) has grown too large.

**Parameters**

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**DidNotConvergeException Class**

```csharp
class Imsl.Stat.DidNotConvergeException : IMSLEntity : ISerializable
```

The iteration did not converge.

---

**Constructors**

**DidNotConvergeException**

```csharp
public DidNotConvergeException()
```

**Description**
The iteration did not converge.

**DidNotConvergeException**

```csharp
public DidNotConvergeException(string message)
```

**Description**
The iteration did not converge.
### DidNotConvergeException

**Description**
The iteration did not converge.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

### DidNotConvergeException


**Description**
The iteration did not converge.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

### DiffObsDeletedException Class

```csharp
public class Imsl.Stat.DiffObsDeletedException : IMSLException : ISerializable
```

Different observations are being deleted from return matrix than were originally entered.

### Constructors

- **DiffObsDeletedException**
  ```csharp
  public DiffObsDeletedException()
  ```
Description
Different observations are being deleted from return matrix than were originally entered.

`DiffObsDeletedException`

`public DiffObsDeletedException(int i)`

Description
Different observations are being deleted from return matrix than were originally entered.

Parameter

   `i` – An `int` which specifies the index of Variance-Covariance matrix.

`DiffObsDeletedException`

`public DiffObsDeletedException(string message)`

Description
Different observations are being deleted from return matrix than were originally entered.

Parameter

   `message` – The error message that explains the reason for the exception.

`DiffObsDeletedException`

`public DiffObsDeletedException(string s, System.Exception exception)`

Description
Different observations are being deleted from return matrix than were originally entered.

Parameters

   `s` – The error message that explains the reason for the exception.

   `exception` – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

`DiffObsDeletedException`


Description
Different observations are being deleted from return matrix than were originally entered.

Parameters

   `info` – The object that holds the serialized object data.

   `context` – The contextual information about the source or destination.

Exceptions

`DiffObsDeletedException` • 1895
EigenvalueException Class

public class Imsl.Stat.EigenvalueException : IMSLException : ISerializable

An error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

Constructors

EigenvalueException

public EigenvalueException()

Description

Eigenvalue error.

EigenvalueException

public EigenvalueException(string message)

Description

Eigenvalue error.

Parameter

message – The error message that explains the reason for the exception.

EigenvalueException

public EigenvalueException(string s, System.Exception exception)

Description

Eigenvalue error.

Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

EigenvalueException


Description

Eigenvalue error.
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

EmptyGroupException Class

```csharp
public class Imsl.Stat.EmptyGroupException : IMSLException : ISerializable
```

There are no observations in a group. Cannot compute statistics.

Constructors

**EmptyGroupException**
```csharp
public EmptyGroupException(int group)
```

**Description**
There are no observations in a group. Cannot compute statistics.

**Parameter**
- group – A int which specifies the index of empty group.

**EmptyGroupException**
```csharp
public EmptyGroupException()
```

**Description**
There are no observations in a group. Cannot compute statistics.

**EmptyGroupException**
```csharp
public EmptyGroupException(string message)
```

**Description**
There are no observations in a group. Cannot compute statistics.

**Parameter**
- message – The error message that explains the reason for the exception.

**EmptyGroupException**
```csharp
public EmptyGroupException(string s, System.Exception exception)
```

**Description**
There are no observations in a group. Cannot compute statistics.

Exceptions

- EmptyGroupException • 1897
Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

EmptyGroupException

EmptyGroupException(System.Runtime.Serialization.SerializationInfo info,

Description
There are no observations in a group. Cannot compute statistics.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

EqConstrInconsistentException Class

public class Imsl.Stat.EqConstrInconsistentException : IMSLException : ISerializable

The equality constraints and the bounds on the variables are found to be inconsistent.

Constructors

EqConstrInconsistentException
public EqConstrInconsistentException()

Description
The equality constraints and the bounds on the variables are found to be inconsistent.

EqConstrInconsistentException
public EqConstrInconsistentException(string message)

Description
The equality constraints and the bounds on the variables are found to be inconsistent.
Parameter

message – The error message that explains the reason for the exception.

**EqConstrInconsistentException**

public EqConstrInconsistentException(string s, System.Exception exception)

**Description**
The equality constraints and the bounds on the variables are found to be inconsistent.

**Parameters**

- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**EqConstrInconsistentException**


**Description**
The equality constraints and the bounds on the variables are found to be inconsistent.

**Parameters**

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**IllConditionedException Class**

public class Imsl.Stat.IllConditionedException : IMSLEception : ISerializable

The problem is ill-conditioned.

**Constructors**

**IllConditionedException**

public IllConditionedException()
Description
The problem is ill-conditioned.

IllConditionedException

public IllConditionedException(string message)

Description
The problem is ill-conditioned.

Parameter
message – The error message that explains the reason for the exception.

IllConditionedException

public IllConditionedException(string s, System.Exception exception)

Description
The problem is ill-conditioned.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

IllConditionedException


Description
The problem is ill-conditioned.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

IncreaseErrRelException Class

public class Imsl.Stat.IncreaseErrRelException : IMSLException : I Serializable

The bound for the relative error is too small.
Constructors

IncreaseErrRelException
public IncreaseErrRelException(double relativeError)

Description
The bound for the relative error is too small.

Parameter
relativeError – A double which specifies the bound for relative error.

IncreaseErrRelException
public IncreaseErrRelException()

Description
The bound for the relative error is too small.

IncreaseErrRelException
public IncreaseErrRelException(string message)

Description
The bound for the relative error is too small.

Parameter
message – The error message that explains the reason for the exception.

IncreaseErrRelException
public IncreaseErrRelException(string s, System.Exception exception)

Description
The bound for the relative error is too small.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

IncreaseErrRelException

Description
The bound for the relative error is too small.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

Exceptions
IncreaseErrRelException • 1901
InitialMAException Class

public class Imsl.Stat.InitialMAException : IMSLException : ISerializable

The initial values for the moving average parameters are not invertable. Execution is halted.

Constructors

InitialMAException
public InitialMAException()

Description
The initial values for the moving average parameters are not invertable. Execution is halted.

InitialMAException
public InitialMAException(string message)

Description
The initial values for the moving average parameters are not invertable. Execution is halted.

Parameter
message – The error message that explains the reason for the exception.

InitialMAException
public InitialMAException(string s, System.Exception exception)

Description
The initial values for the moving average parameters are not invertable. Execution is halted.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

InitialMAException

Description
The initial values for the moving average parameters are not invertable. Execution is halted.
InvalidMatrixException Class

public class Imsl.Stat.InvalidMatrixException : IMSLException : ISerializable

Exception thrown if a computed correlation is greater than one for some pair of variables.

Constructor

InvalidMatrixException

public InvalidMatrixException(int var1, int var2)

Description

Creates an InvalidMatrixException thrown if a computed correlation is greater than one for some pair of variables.

Parameters

var1 – is the index of the first variable in the pair.
var2 – is the index of the second variable in the pair.

InvalidPartialCorrelationException Class

public class Imsl.Stat.InvalidPartialCorrelationException : IMSLException : ISerializable

Exception thrown if a computed partial correlation is greater than one for some pair of variables.

Constructor

InvalidPartialCorrelationException

public InvalidPartialCorrelationException(int var1, int var2)
Description
Creates an InvalidPartialCorrelationException thrown if a computed partial correlation is greater than one for some pair of variables.

Parameters

- var1 – is the index of the first variable in the pair.
- var2 – is the index of the second variable in the pair.

MatrixSingularException Class

```csharp
public class Imsl.Stat.MatrixSingularException : IMSLException : ISerializable

The input matrix is singular.
```

Constructors

- **MatrixSingularException**
  ```csharp
  public MatrixSingularException()
  ```
  Description
  The input matrix is singular.

- **MatrixSingularException**
  ```csharp
  public MatrixSingularException(string message)
  ```
  Description
  The input matrix is singular.
  Parameter
  message – The error message that explains the reason for the exception.

- **MatrixSingularException**
  ```csharp
  public MatrixSingularException(string s, System.Exception exception)
  ```
  Description
  The input matrix is singular.
Parameters

- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

MatrixSingularException

Description
The input matrix is singular.

Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

MoreObsDelThanEnteredException Class

public class Imsl.Stat.MoreObsDelThanEnteredException : IMSLEException : ISerializable

More observations are being deleted from the output covariance matrix than were originally entered (the corresponding row, column of the incidence matrix is less than zero).

Constructors

MoreObsDelThanEnteredException

public MoreObsDelThanEnteredException(int j, int k)

Description
More observations are being deleted from the output covariance matrix than were originally entered (the corresponding row, column of the incidence matrix is less than zero).

Parameters

- j – A int which specifies the row index of Variance-Covariance matrix.
- k – A int which specifies the column index of Variance-Covariance matrix.

MoreObsDelThanEnteredException

public MoreObsDelThanEnteredException()
Description
More observations are being deleted from the output covariance matrix than were originally entered (the corresponding row, column of the incidence matrix is less than zero).

MoreObsDelThanEnteredException
public MoreObsDelThanEnteredException(string message)

Description
More observations are being deleted from the output covariance matrix than were originally entered (the corresponding row, column of the incidence matrix is less than zero).

Parameter
message – The error message that explains the reason for the exception.

MoreObsDelThanEnteredException
public MoreObsDelThanEnteredException(string s, System.Exception exception)

Description
More observations are being deleted from the output covariance matrix than were originally entered (the corresponding row, column of the incidence matrix is less than zero).

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

MoreObsDelThanEnteredException

Description
More observations are being deleted from the output covariance matrix than were originally entered (the corresponding row, column of the incidence matrix is less than zero).

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

NegativeFreqException Class

public class Imsl.Stat.NegativeFreqException : IMSLException : ISerializable
A negative frequency was encountered.
Constructors

**NegativeFreqException**  
public NegativeFreqException(int rowIndex, int invocation, double val)

**Description**  
A negative frequency was encountered.

**Parameters**
  - **rowIndex** – An int which specifies the row index of X for which the frequency is negative.
  - **invocation** – An int which specifies the invocation number at which the error occurred. A 3 would indicate that the error occurred on the third invocation.
  - **val** – A double which represents the value of the frequency encountered.

**NegativeFreqException**  
public NegativeFreqException()

**Description**  
A negative frequency was encountered.

**NegativeFreqException**  
public NegativeFreqException(string message)

**Description**  
A negative frequency was encountered.

**Parameter**
  - **message** – The error message that explains the reason for the exception.

**NegativeFreqException**  
public NegativeFreqException(string s, System.Exception exception)

**Description**  
A negative frequency was encountered.

**Parameters**
  - **s** – The error message that explains the reason for the exception.
  - **exception** – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**NegativeFreqException**  
NegativeFreqException(System.Runtime.Serialization.SerializationInfo info,  

**Description**  
A negative frequency was encountered.

**Exceptions**  
NegativeFreqException ● 1907
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**NegativeWeightException Class**

```csharp
public class Imsl.Stat.NegativeWeightException : IMSLException : ISerializable
```

A negative weight was encountered.

---

**Constructors**

**NegativeWeightException**

```csharp
public NegativeWeightException(int rowIndex, int invocation, double val)
```

**Description**

A negative weight was encountered.

**Parameters**

- `rowIndex` – An `int` which specifies the row index of X for which the weight is negative.
- `invocation` – An `int` which specifies the invocation number at which the error occurred. A 3 would indicate that the error occurred on the third invocation.
- `val` – An `double` which represents the value of the weight encountered.

**NegativeWeightException**

```csharp
public NegativeWeightException()
```

**Description**

A negative weight was encountered.

**NegativeWeightException**

```csharp
public NegativeWeightException(string message)
```

**Description**

A negative weight was encountered.

**Parameter**

- `message` – The error message that explains the reason for the exception.

**NegativeWeightException**

```csharp
public NegativeWeightException(string s, System.Exception exception)
```
Description
A negative weight was encountered.

Parameters
- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NegativeWeightException

Description
A negative weight was encountered.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

NewInitialGuessException Class

public class Imsl.Stat.NewInitialGuessException : IMSLException : ISerializable

The iteration has not made good progress.

Constructors

NewInitialGuessException
public NewInitialGuessException()

Description
The iteration has not made good progress.

NewInitialGuessException
public NewInitialGuessException(string message)

Description
The iteration has not made good progress.
Parameter

message – The error message that explains the reason for the exception.

NewInitialGuessException

public NewInitialGuessException(string s, System.Exception exception)

Description
The iteration has not made good progress.

Parameters

s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NewInitialGuessException


Description
The iteration has not made good progress.

Parameters

info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

NoAcceptableModelFoundException Class

public class Imsl.Stat.NoAcceptableModelFoundException : IMSLException : ISerializable

No appropriate ARIMA model could be found.

Constructors

NoAcceptableModelFoundException

public NoAcceptableModelFoundException()
No appropriate ARIMA model could be found.

**NoAcceptableModelFoundException**

```java
public NoAcceptableModelFoundException(string message)
```

**Description**
No appropriate ARIMA model could be found.

**Parameter**
- `message` – The error message that explains the reason for the exception.

```java
public NoAcceptableModelFoundException(string s, System.Exception exception)
```

**Description**
No appropriate ARIMA model could be found.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```java
```

**Description**
No appropriate ARIMA model could be found.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**NoConvergenceException Class**

```java
public class Imsl.Stat.NoConvergenceException : IMSLException : ISerializable
```

Convergence did not occur within the maximum number of iterations.

---

**Exceptions**

NoConvergenceException • 1911
Constructors

**NoConvergenceException**

```java
public NoConvergenceException(int maximumIterations)
```

**Description**
Convergence did not occur within the maximum number of iterations.

**Parameter**
- `maximumIterations` – A `int` which specifies the maximum number of iterations allowed.

**NoConvergenceException**

```java
public NoConvergenceException()
```

**Description**
Convergence did not occur within the maximum number of iterations.

**NoConvergenceException**

```java
public NoConvergenceException(string message)
```

**Description**
Convergence did not occur within the maximum number of iterations.

**Parameter**
- `message` – The error message that explains the reason for the exception.

**NoConvergenceException**

```java
public NoConvergenceException(string s, System.Exception exception)
```

**Description**
Convergence did not occur within the maximum number of iterations.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**NoConvergenceException**

```java
NoConvergenceException(System.Runtime.Serialization.SerializationInfo info,
```

**Description**
Convergence did not occur within the maximum number of iterations.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.
NoDegreesOfFreedomException Class

public class Imsl.Stat.NoDegreesOfFreedomException : IMSLException : ISerializable

No degrees of freedom error.

Constructors

NoDegreesOfFreedomException

public NoDegreesOfFreedomException(int nvar, int nf)

Description
No degrees of freedom error.

Parameters

nvar – A int which specifies the number of variables.

nf – A int which specifies the number of factors.

NoDegreesOfFreedomException

public NoDegreesOfFreedomException()

Description
No degrees of freedom error.

NoDegreesOfFreedomException

public NoDegreesOfFreedomException(string message)

Description
No degrees of freedom error.

Parameter

message – The error message that explains the reason for the exception.

NoDegreesOfFreedomException

public NoDegreesOfFreedomException(string s, System.Exception exception)

Description
No degrees of freedom error.

Exceptions

NoDegreesOfFreedomException • 1913
Parameters

- The error message that explains the reason for the exception.
- The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NoDegreesOfFreedomException

Description
No degrees of freedom error.

Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

NoProgressException Class

public class Imsl.Stat.NoProgressException : IMSLException : ISerializable

The algorithm is not making any progress. Try a new initial guess.

Constructors

NoProgressException
public NoProgressException()

Description
The algorithm is not making any progress. Try a new initial guess.

NoProgressException
public NoProgressException(string message)

Description
The algorithm is not making any progress. Try a new initial guess.

Parameter

- message – The error message that explains the reason for the exception.

NoProgressException
public NoProgressException(string s, System.Exception exception)
Description
The algorithm is not making any progress. Try a new initial guess.

Parameters
- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NoProgressException

Description
The algorithm is not making any progress. Try a new initial guess.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

NoVariationInputException Class

public class Imsl.Stat.NoVariationInputException : IMSLException :
ISerializable
There is no variation in the input data.

Constructors

NoVariationInputException
public NoVariationInputException()  

Description
There is no variation in the input data.

NoVariationInputException
public NoVariationInputException(string message)

Description
There is no variation in the input data.

Exceptions
NoVariationInputException • 1915
Parameter
	message – The error message that explains the reason for the exception.

**NoVariationInputException**

```csharp
public NoVariationInputException(string s, System.Exception exception)
```

**Description**

There is no variation in the input data.

**Parameters**

- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**NoVariationInputException**

```csharp
```

**Description**

There is no variation in the input data.

**Parameters**

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

**NoVectorXException Class**

```csharp
public class Imsl.Stat.NoVectorXException : IMSLException : ISerializable
```

No vector X satisfies all of the constraints.

**Constructors**

**NoVectorXException**

```csharp
public NoVectorXException()
```

**Description**

No vector X satisfies all of the constraints.

**NoVectorXException**

```csharp
public NoVectorXException(string message)
```

**1916 • NoVectorXException**
No vector X satisfies all of the constraints.

**Parameters**
- **message** – The error message that explains the reason for the exception.

### NoVectorXException

```csharp
public NoVectorXException(string s, System.Exception exception)
```

**Description**
No vector X satisfies all of the constraints.

**Parameters**
- **s** – The error message that explains the reason for the exception.
- **exception** – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

### NoVectorXException

```csharp
```

**Description**
No vector X satisfies all of the constraints.

**Parameters**
- **info** – The object that holds the serialized object data.
- **context** – The contextual information about the source or destination.

### NonInvertibleException Class

```csharp
public class Imsl.Stat.NonInvertibleException : IMSLException : ISerializable
```

The solution is noninvertible.

### Constructors

```csharp
NonInvertibleException()
```

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonInvertibleException</td>
<td>The solution is noninvertible.</td>
</tr>
<tr>
<td>1917</td>
<td></td>
</tr>
</tbody>
</table>
Description
The solution is noninvertible.

**NonInvertibleException**
public NonInvertibleException(string message)

Description
The solution is noninvertible.

Parameter
message – The error message that explains the reason for the exception.

**NonInvertibleException**
public NonInvertibleException(string s, System.Exception exception)

Description
The solution is noninvertible.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**NonInvertibleException**

Description
The solution is noninvertible.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

---

**NonPosVarianceException Class**

public class Imsl.Stat.NonPosVarianceException : IMSLException : ISerializable

The problem is ill-conditioned.
Constructors

NonPosVarianceException
public NonPosVarianceException(double var)

Description
The problem is ill-conditioned.

Parameter
var – A double which specifies the variance.

NonPosVarianceException
public NonPosVarianceException()

Description
The problem is ill-conditioned.

NonPosVarianceException
public NonPosVarianceException(string message)

Description
The problem is ill-conditioned.

Parameter
message – The error message that explains the reason for the exception.

NonPosVarianceException
public NonPosVarianceException(string s, System.Exception exception)

Description
The problem is ill-conditioned.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NonPosVarianceException

Description
The problem is ill-conditioned.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

Exceptions
NonPosVarianceException • 1919
NonPosVarianceXYException Class

public class Imsl.Stat.NonPosVarianceXYException : IMSLException : ISerializable

The problem is ill-conditioned.

Constructors

NonPosVarianceXYException
public NonPosVarianceXYException(string varName, double var)

Description
The problem is ill-conditioned.

Parameters

  varName – A string which specifies either “X” or “Y”.
  var – A double which specifies the variance.

NonPosVarianceXYException
public NonPosVarianceXYException()

Description
The problem is ill-conditioned.

NonPosVarianceXYException
public NonPosVarianceXYException(string message)

Description
The problem is ill-conditioned.

Parameter

  message – The error message that explains the reason for the exception.

NonPosVarianceXYException
public NonPosVarianceXYException(string s, System.Exception exception)

Description
The problem is ill-conditioned.
Parameters
  s – The error message that explains the reason for the exception.
  exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NonPosVarianceXYException
NonPosVarianceXYException(System.Runtime.Serialization.SerializationInfo info,

Description
The problem is ill-conditioned.

Parameters
  info – The object that holds the serialized object data.
  context – The contextual information about the source or destination.

NonPositiveEigenvalueException Class

public class Imsl.Stat.NonPositiveEigenvalueException : IMSLException : ISerializable
Maximum number of iterations exceeded.

Constructors

NonPositiveEigenvalueException
public NonPositiveEigenvalueException(int iter, int i, double eval)

Description
Maximum number of iterations exceeded.

Parameters
  iter – A int which specifies the iteration number.
  i – A int which specifies the eigenvalue index.
  eval – A double which specifies the eigenvalue.

NonPositiveEigenvalueException
public NonPositiveEigenvalueException()
Maximum number of iterations exceeded.

**NonPositiveEigenvalueException**

```csharp
public NonPositiveEigenvalueException(string message)
```

**Description**

Maximum number of iterations exceeded.

**Parameter**

- `message` – The error message that explains the reason for the exception.

```csharp
public NonPositiveEigenvalueException(string s, System.Exception exception)
```

**Description**

Maximum number of iterations exceeded.

**Parameters**

- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```csharp
```

**Description**

Maximum number of iterations exceeded.

**Parameters**

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**NonStationaryException Class**

```csharp
public class Imsl.Stat.NonStationaryException : IMSLException : ISerializable

The solution is nonstationary.
```
Constructors

**NonStationaryException**

```java
public NonStationaryException()
```

**Description**
The solution is nonstationary.

```java
public NonStationaryException(string message)
```

**Description**
The solution is nonstationary.

**Parameter**

message – The error message that explains the reason for the exception.

```java
public NonStationaryException(string s, System.Exception exception)
```

**Description**
The solution is nonstationary.

**Parameters**

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```java
public NonStationaryException(System.Runtime.Serialization.SerializationInfo info,
```

**Description**
The solution is nonstationary.

**Parameters**

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

---

**NoPositiveVarianceException Class**

```java
public class Imsl.Stat.NoPositiveVarianceException : IMSLEException : ISerializable
```

**Exceptions**

NoPositiveVarianceException • 1923
No variable has positive variance. The Mahalanobis distances cannot be computed.

**Constructors**

**NoPositiveVarianceException**

```csharp
public NoPositiveVarianceException()
```

**Description**

No variable has positive variance. The Mahalanobis distances cannot be computed.

**NoPositiveVarianceException**

```csharp
public NoPositiveVarianceException(string message)
```

**Description**

No variable has positive variance. The Mahalanobis distances cannot be computed.

**Parameter**

- `message` – The error message that explains the reason for the exception.

**NoPositiveVarianceException**

```csharp
public NoPositiveVarianceException(string s, System.Exception exception)
```

**Description**

No variable has positive variance. The Mahalanobis distances cannot be computed.

**Parameters**

- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**NoPositiveVarianceException**

```csharp
```

**Description**

No variable has positive variance. The Mahalanobis distances cannot be computed.

**Parameters**

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.
**NotCDFException Class**

```java
public class Imsl.Stat.NotCDFException : IMSLException : ISerializable
```

The function is not a Cumulative Distribution Function (CDF).

**Constructors**

**NotCDFException**
```java
public NotCDFException(double lowerBound, double upperBound)
```

**Description**
The function is not a Cumulative Distribution Function (CDF).

**Parameters**
- `lowerBound` – A `double` containing the lower bound to be displayed in message.
- `upperBound` – A `double` containing the upper bound to be displayed in message.

**NotCDFException**
```java
public NotCDFException(double range)
```

**Description**
The function is not a Cumulative Distribution Function (CDF).

**Parameter**
- `range` – A `double` containing the probability of the range.

**NotCDFException**
```java
public NotCDFException(double x1, double x2, double f1)
```

**Description**
The function is not a Cumulative Distribution Function (CDF).

**Parameters**
- `x1` – is the first point
- `x2` – is the second point
- `f1` – is the common value for F(x1) and F(x2)

**Remarks**
The CDF function is not monotone, F(x1) = F(x2). No unique inverse exists.

**NotCDFException**
```java
public NotCDFException(double lowerBound, double upperBound, double xx, int i)
```

**Exceptions**

- **NotCDFException** • 1925
Description
The function is not a Cumulative Distribution Function (CDF).

Parameters
- lowerBound – A double containing the lower bound for the CDF value.
- upperBound – A double containing the upper bound for the CDF value.
- xx – A double containing the value at a cutpoint.
- i – The index of the cutpoint that is out of range.

Remarks
The cdf function is not a cumulative distribution function because its value at a cutpoint is out of the expected range, [plower, ppupper].

NotCDFException
public NotCDFException()

Description
The function is not a Cumulative Distribution Function (CDF).

NotCDFException
public NotCDFException(string message)

Description
The function is not a Cumulative Distribution Function (CDF).

Parameter
- message – The error message that explains the reason for the exception.

NotCDFException
public NotCDFException(string s, System.Exception exception)

Description
The function is not a Cumulative Distribution Function (CDF).

Parameters
- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NotCDFException

Description
The function is not a Cumulative Distribution Function (CDF).
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

**NotPositiveDefiniteException Class**

```java
public class Imsl.Stat.NotPositiveDefiniteException : IMSLEException : ISerializable

Covariance matrix is not positive definite.
```

**Constructors**

**NotPositiveDefiniteException**
```java
public NotPositiveDefiniteException(int i)
```

**Description**
Covariance matrix is not positive definite.

**Parameter**
- i – Variable i is linearly related to the other variables in the factor analysis.

**NotPositiveDefiniteException**
```java
public NotPositiveDefiniteException()
```

**Description**
Covariance matrix is not positive definite.

**NotPositiveDefiniteException**
```java
public NotPositiveDefiniteException(string message)
```

**Description**
Covariance matrix is not positive definite.

**Parameter**
- message – The error message that explains the reason for the exception.

**NotPositiveDefiniteException**
```java
public NotPositiveDefiniteException(string s, System.Exception exception)
```

**Exceptions**

- NotPositiveDefiniteException
- IMSLEException
- ISerializable
Description
Covariance matrix is not positive definite.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException
parameter is not a null reference, the current exception is raised in a catch block that handles the
inner exception.

NotPositiveDefiniteException
NotPositiveDefiniteException(System.Runtime.Serialization.SerializationInfo

Description
Covariance matrix is not positive definite.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

NotPositiveSemiDefiniteException Class

public class Imsl.Stat.NotPositiveSemiDefiniteException : IMSLException : ISerializable
Covariance matrix is not positive semi-definite.

Constructors

NotPositiveSemiDefiniteException
public NotPositiveSemiDefiniteException()

Description
Covariance matrix is not positive semi-definite.

NotPositiveSemiDefiniteException
public NotPositiveSemiDefiniteException(string message)

Description
Covariance matrix is not positive semi-definite.
Parameter

message – The error message that explains the reason for the exception.

NotPositiveSemiDefiniteException

public NotPositiveSemiDefiniteException(string s, System.Exception exception)

Description
Covariance matrix is not positive semi-definite.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NotPositiveSemiDefiniteException


Description
Covariance matrix is not positive semi-definite.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

NotSemiDefiniteException Class

public class Imsl.Stat.NotSemiDefiniteException : IMSLException : ISerializable

Hessian matrix is not semi-definite.

Constructors

NotSemiDefiniteException

public NotSemiDefiniteException()
Description
Hessian matrix is not semi-definite.

NotNullSemiDefiniteException
public NotSemiDefiniteException(string message)

Description
Hessian matrix is not semi-definite.

Parameter
message – The error message that explains the reason for the exception.

NotNullSemiDefiniteException
public NotSemiDefiniteException(string s, System.Exception exception)

Description
Hessian matrix is not semi-definite.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException
parameter is not a null reference, the current exception is raised in a catch block that handles the
inner exception.

NotNullSemiDefiniteException
NotSemiDefiniteException(System.Runtime.Serialization.SerializationInfo info,

Description
Hessian matrix is not semi-definite.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

NoVariablesEnteredException Class

public class Imsl.Stat.NoVariablesEnteredException : IMSLException :
ISerializable

No Variables can enter the model.
Constructors

**NoVariablesEnteredException**

```java
public NoVariablesEnteredException()
```

**Description**
No Variables can enter the model.

**NoVariablesEnteredException**

```java
public NoVariablesEnteredException(string message)
```

**Description**
No Variables can enter the model.

**Parameter**

- `message` – The error message that explains the reason for the exception.

**NoVariablesEnteredException**

```java
public NoVariablesEnteredException(string s, System.Exception exception)
```

**Description**
No Variables can enter the model.

**Parameters**

- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**NoVariablesEnteredException**

```java
```

**Description**
No Variables can enter the model.

**Parameters**

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**NoVariablesException Class**

```java
public class Imsl.Stat.NoVariablesException : IMSLEException : ISerializable
```

No variables can enter the model.

---

**Exceptions**

- **NoVariablesException** – 1931
Constructors

NoVariablesException
public NoVariablesException()

Description
No variables can enter the model.

NoVariablesException
public NoVariablesException(string message)

Description
No variables can enter the model.

Parameter
message – The error message that explains the reason for the exception.

NoVariablesException
public NoVariablesException(string s, System.Exception exception)

Description
No variables can enter the model.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NoVariablesException

Description
No variables can enter the model.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

NoVariationInputException Class

public class Imsl.Stat.NoVariationInputException : IMSLException : ISerializable

1932 • NoVariationInputException  IMSL C# Numerical Library
Constructors

NoVariationInputException
public NoVariationInputException()

Description
There is no variation in the input data.

NoVariationInputException
public NoVariationInputException(string message)

Description
There is no variation in the input data.

Parameter

message – The error message that explains the reason for the exception.

NoVariationInputException
public NoVariationInputException(string s, System.Exception exception)

Description
There is no variation in the input data.

Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

NoVariationInputException

Description
There is no variation in the input data.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.
NoVectorXException Class

```csharp
public class Imsl.Stat.NoVectorXException : IMSLException : ISerializable
No vector X satisfies all of the constraints.
```

**Constructors**

**NoVectorXException()**
```csharp
public NoVectorXException()
```

**Description**
No vector X satisfies all of the constraints.

**NoVectorXException(string message)**
```csharp
public NoVectorXException(string message)
```

**Description**
No vector X satisfies all of the constraints.

**Parameter**
message – The error message that explains the reason for the exception.

**NoVectorXException(string s, System.Exception exception)**
```csharp
public NoVectorXException(string s, System.Exception exception)
```

**Description**
No vector X satisfies all of the constraints.

**Parameters**
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```csharp
```

**Description**
No vector X satisfies all of the constraints.
Parameters

- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**PooledCovarianceSingularException Class**

```java
public class Imsl.Stat.PooledCovarianceSingularException : IMSLException : ISerializable

The pooled variance-Covariance matrix is singular.
```

**Constructors**

**PooledCovarianceSingularException**

```java
public PooledCovarianceSingularException()
```

**Description**

The pooled variance-Covariance matrix is singular.

**PooledCovarianceSingularException**

```java
public PooledCovarianceSingularException(string message)
```

**Description**

The pooled variance-Covariance matrix is singular.

**Parameter**

- message – The error message that explains the reason for the exception.

**PooledCovarianceSingularException**

```java
public PooledCovarianceSingularException(string s, System.Exception exception)
```

**Description**

The pooled variance-Covariance matrix is singular.

**Parameters**

- s – The error message that explains the reason for the exception.
- exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

---

**Exceptions**

PooledCovarianceSingularException • 1935
PooledCovarianceSingularException


**Description**
The pooled variance-Covariance matrix is singular.

**Parameters**
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

---

**RankException Class**

```csharp
public class Imsl.Stat.RankException : IMSLException : ISerializable
```

Rank of covariance matrix error.

---

**Constructors**

**RankException**

```csharp
public RankException(int rank, int nf)
```

**Description**
Rank of covariance matrix error.

**Parameters**
- rank – An int which specifies the rank of the covariance matrix.
- nf – An int which specifies the number of factors.

**RankException**

```csharp
public RankException()
```

**Description**
Rank of covariance matrix error.

**RankException**

```csharp
public RankException(string message)
```

**Description**
Rank of covariance matrix error.
Parameter

message – The error message that explains the reason for the exception.

RankException

public RankException(string s, System.Exception exception)

Description

Rank of covariance matrix error.

Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

RankException


Description

Rank of covariance matrix error.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

RankDeficientException Class

public class Imsl.Stat.RankDeficientException : IMSLEexception : ISerializable

The model has been determined to be rank deficient.

Constructors

RankDeficientException

public RankDeficientException(int rank)

Description

The model has been determined to be rank deficient.

Exceptions

RankDeficientException ● 1937
Parameter

rank – An int which specifies the rank of the model.

__RankDeficientException__

```csharp
public RankDeficientException()
```

Description

The model has been determined to be rank deficient.

```csharp
public RankDeficientException(string message)
```

Description

The model has been determined to be rank deficient.

Parameter

message – The error message that explains the reason for the exception.

```csharp
public RankDeficientException(string s, System.Exception exception)
```

Description

The model has been determined to be rank deficient.

Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```csharp
RankDeficientException(System.Runtime.Serialization.SerializationInfo info,
```

Description

The model has been determined to be rank deficient.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

---

**ScaleFactorZeroException Class**

```csharp
public class Imsl.Stat.ScaleFactorZeroException : IMSLException : ISerializable
```

1938 • ScaleFactorZeroException  IMSL C# Numerical Library
The computations cannot continue because a scale factor is zero.

**Constructors**

**ScaleFactorZeroException**  
public ScaleFactorZeroException(int index)

**Description**  
The computations cannot continue because a scale factor is zero.

**Parameter**  
index – An int which specifies the index of the scale factor array at which scale factor is zero.

**ScaleFactorZeroException**  
public ScaleFactorZeroException()

**Description**  
The computations cannot continue because a scale factor is zero.

**ScaleFactorZeroException**  
public ScaleFactorZeroException(string message)

**Description**  
The computations cannot continue because a scale factor is zero.

**Parameter**  
message – The error message that explains the reason for the exception.

**ScaleFactorZeroException**  
public ScaleFactorZeroException(string s, System.Exception exception)

**Description**  
The computations cannot continue because a scale factor is zero.

**Parameters**  
s – The error message that explains the reason for the exception.

**Parameters**  
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**ScaleFactorZeroException**  
ScaleFactorZeroException(System.Runtime.Serialization.SerializationInfo info,  

**Description**  
The computations cannot continue because a scale factor is zero.

**Exceptions**  
ScaleFactorZeroException ● 1939
Parameters

- **info** – The object that holds the serialized object data.
- **context** – The contextual information about the source or destination.

---

**SingularException Class**

```csharp
public class Imsl.Stat.SingularException : IMSLException : ISerializable

Covariance matrix is singular.
```

**Constructors**

**SingularException**

```csharp
public SingularException(int i)
```

**Description**

Covariance matrix is singular.

**Parameter**

- `i` – Variable i is linearly related to the other variables.

**SingularException**

```csharp
public SingularException()
```

**Description**

Covariance matrix is singular.

**SingularException**

```csharp
public SingularException(string message)
```

**Description**

Covariance matrix is singular.

**Parameter**

- `message` – The error message that explains the reason for the exception.

**SingularException**

```csharp
public SingularException(string s, System.Exception exception)
```

**Description**

Covariance matrix is singular.
Parameters

s – The error message that explains the reason for the exception.

t.exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

SingularException

SingularException(System.Runtime.Serialization.SerializationInfo info,

Description
Covariance matrix is singular.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

SingularTriangularMatrixException Class

public class Imsl.Stat.SingularTriangularMatrixException : IMSLException : ISerializable

Triangular matrix is singular.

Constructors

SingularTriangularMatrixException

public SingularTriangularMatrixException(int i)

Description
Triangular matrix is singular.

Parameter

i – Variable i is the index of the first zero diagonal element.

SingularTriangularMatrixException

public SingularTriangularMatrixException()
Description
Triangular matrix is singular.

**SingularTriangularMatrixException**

```csharp
public SingularTriangularMatrixException(string message)
```

**Description**
Triangular matrix is singular.

**Parameter**

- *message* – The error message that explains the reason for the exception.

```csharp
public SingularTriangularMatrixException(string s, System.Exception exception)
```

**Description**
Triangular matrix is singular.

**Parameters**

- *s* – The error message that explains the reason for the exception.
- *exception* – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

```csharp
```

**Description**
Triangular matrix is singular.

**Parameters**

- *info* – The object that holds the serialized object data.
- *context* – The contextual information about the source or destination.

---

**SumOfWeightsNegException Class**

```csharp
public class Imsl.Stat.SumOfWeightsNegException : IMSLException : ISerializable
```

The sum of the weights have become negative.
Constructors

**SumOfWeightsNegException**
public SumOfWeightsNegException(int group)

**Description**
The sum of the weights have become negative.

**Parameter**

  group – An int which specifies the group for which the sum of the weights have become negative.

**SumOfWeightsNegException**
public SumOfWeightsNegException()

**Description**
The sum of the weights have become negative.

**SumOfWeightsNegException**
public SumOfWeightsNegException(string message)

**Description**
The sum of the weights have become negative.

**Parameter**

  message – The error message that explains the reason for the exception. A String containing the error message.

**SumOfWeightsNegException**
public SumOfWeightsNegException(string s, System.Exception exception)

**Description**
The sum of the weights have become negative.

**Parameters**

  s – The error message that explains the reason for the exception.

  exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**SumOfWeightsNegException**

**Description**
The sum of the weights have become negative.

Exceptions

  SumOfWeightsNegException • 1943
Parameters

- **info** – The object that holds the serialized object data.
- **context** – The contextual information about the source or destination.

---

**TooManyCallsException Class**

```csharp
public class Imsl.Stat.TooManyCallsException : IMSLException : ISerializable

The number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.
```

**Constructors**

**TooManyCallsException**

```csharp
public TooManyCallsException()
```

**Description**

The number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.

**TooManyCallsException**

```csharp
public TooManyCallsException(string message)
```

**Description**

The number of calls to the function has exceeded the maximum number of iterations times the number of moving average (MA) parameters + 1.

**Parameter**

- **message** – The error message that explains the reason for the exception.

**TooManyCallsException**

```csharp
public TooManyCallsException(string s, System.Exception exception)
```

**Description**

The number of calls to the function has exceeded the maximum number of iterations.

**Parameters**

- **s** – The error message that explains the reason for the exception.
- **exception** – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.
TooManyCallsException


Description
The number of calls to the function has exceeded the maximum number of iterations.

Parameters
- info – The object that holds the serialized object data.
- context – The contextual information about the source or destination.

TooManyFunctionEvaluationsException Class

public class Imsl.Stat.TooManyFunctionEvaluationsException : IMSLException : ISerializable
Maximum number of function evaluations exceeded.

Constructors

TooManyFunctionEvaluationsException
public TooManyFunctionEvaluationsException(int maximumNumberOfEvaluations)

Description
Maximum number of function evaluations exceeded.

Parameter
- maximumNumberOfEvaluations – A int which specifies the maximum number of function evaluations allowed.

TooManyFunctionEvaluationsException
public TooManyFunctionEvaluationsException()

Description
Maximum number of function evaluations exceeded.

TooManyFunctionEvaluationsException
public TooManyFunctionEvaluationsException(string message)

Description
Maximum number of function evaluations exceeded.

Exceptions
TooManyFunctionEvaluationsException • 1945
Parameter

message – The error message that explains the reason for the exception.

**TooManyFunctionEvaluationsException**

```csharp
public TooManyFunctionEvaluationsException(string s, System.Exception exception)
```

**Description**

Maximum number of function evaluations exceeded.

**Parameters**

- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**TooManyFunctionEvaluationsException**

```csharp
```

**Description**

Maximum number of function evaluations exceeded.

**Parameters**

- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.

---

**TooManyIterationsException Class**

```csharp
public class Imsl.Stat.TooManyIterationsException : IMSLException : ISerializable
```

Maximum number of iterations exceeded.

**Constructors**

**TooManyIterationsException**

```csharp
public TooManyIterationsException(int maximumNumberOfIterations)
```

**Description**

Maximum number of iterations exceeded.
Parameter

maximumNumberOfIterations – A int which specifies the maximum number of iterations allowed.

TooManyIterationsException

public TooManyIterationsException()

Description
Maximum number of iterations exceeded.

TooManyIterationsException

public TooManyIterationsException(string message)

Description
Maximum number of iterations exceeded.

Parameter

message – The error message that explains the reason for the exception.

TooManyIterationsException

public TooManyIterationsException(string s, System.Exception exception)

Description
Maximum number of iterations exceeded.

Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

TooManyIterationsException


Description
Maximum number of iterations exceeded.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.
TooManyIterationsReTryException Class

public class Imsl.Stat.TooManyIterationsReTryException : IMSLException : ISerializable

The maximum number of iterations was exceeded, increase maximum iterations or try a different parameter estimation method.

Constructors

TooManyIterationsReTryException
public TooManyIterationsReTryException(int maximumNumberOfIterations)

Description
The maximum number of iterations was exceeded, increase maximum iterations or try a different parameter estimation method.

Parameter
maximumNumberOfIterations – A int which specifies the maximum number of iterations allowed.

TooManyIterationsReTryException
public TooManyIterationsReTryException(string message)

Description
The maximum number of iterations was exceeded, increase maximum iterations or try a different parameter estimation method.

Parameter
message – The error message that explains the reason for the exception.

TooManyIterationsReTryException
public TooManyIterationsReTryException(string s, System.Exception exception)

Description
The maximum number of iterations was exceeded, increase maximum iterations or try a different parameter estimation method.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.
TooManyIterationsReTryException

Description
The maximum number of iterations was exceeded, increase maximum iterations or try a different parameter estimation method.

Parameters
- info - The object that holds the serialized object data.
- context - The contextual information about the source or destination.

TooManyJacobianEvalException Class

public class Imsl.Stat.TooManyJacobianEvalException : IMSLException : ISerializable

Maximum number of Jacobian evaluations exceeded.

Constructors

TooManyJacobianEvalException
public TooManyJacobianEvalException()

Description
Maximum number of Jacobian evaluations exceeded.

TooManyJacobianEvalException
public TooManyJacobianEvalException(string message)

Description
Maximum number of Jacobian evaluations exceeded.

Parameter
- message - The error message that explains the reason for the exception.

TooManyJacobianEvalException
public TooManyJacobianEvalException(string s, System.Exception exception)

Description
Maximum number of Jacobian evaluations exceeded.

Exceptions TooManyJacobianEvalException • 1949
Parameters

s – The error message that explains the reason for the exception.

exception – The exception that is the cause of the current exception. If the innerException
parameter is not a null reference, the current exception is raised in a catch block that handles the
inner exception.

TooManyJacobianEvalException

TooManyJacobianEvalException(System.Runtime.Serialization.SerializationInfo

Description
Maximum number of Jacobian evaluations exceeded.

Parameters

info – The object that holds the serialized object data.

context – The contextual information about the source or destination.

TooManyObsDeletedException Class

public class Imsl.Stat.TooManyObsDeletedException : IMSLException :
ISerializable

More observations have been deleted than were originally entered (the sum of frequencies has become
negative).

Constructors

TooManyObsDeletedException

public TooManyObsDeletedException()

Description
More observations have been deleted than were originally entered (the sum of frequencies has become
negative).

TooManyObsDeletedException

public TooManyObsDeletedException(string message)

Description
More observations have been deleted than were originally entered (the sum of frequencies has become
negative).
Parameter
message – The error message that explains the reason for the exception.

TooManyObsDeletedException
public TooManyObsDeletedException(string s, System.Exception exception)

Description
More observations have been deleted than were originally entered (the sum of frequencies has become negative).

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

TooManyObsDeletedException

Description
More observations have been deleted than were originally entered (the sum of frequencies has become negative).

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

VarsDeterminedException Class

public class Imsl.Stat.VarsDeterminedException : IMSLException : I Serializable
The variables are determined by the equality constraints.

Constructors

VarsDeterminedException
public VarsDeterminedException()

Exceptions
VarsDeterminedException • 1951
Description
The variables are determined by the equality constraints.

**VarsDeterminedException**
public VarsDeterminedException(string message)

Description
The variables are determined by the equality constraints.

Parameter
message – The error message that explains the reason for the exception.

**VarsDeterminedException**
public VarsDeterminedException(string s, System.Exception exception)

Description
The variables are determined by the equality constraints.

Parameters
s – The error message that explains the reason for the exception.
exception – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**VarsDeterminedException**

Description
The variables are determined by the equality constraints.

Parameters
info – The object that holds the serialized object data.
context – The contextual information about the source or destination.

---

**ZeroNormException Class**

public class Imsl.Stat.ZeroNormException : IMSLException : ISerializable

The computations cannot continue because the Euclidean norm of the column is equal to zero.
Constructors

**ZeroNormException**

```java
public ZeroNormException(int index)
```

**Description**
The computations cannot continue because the Euclidean norm of the column is equal to zero.

**Parameter**
- `index` – An int which specifies the column index for which the norm has been found to be zero.

**ZeroNormException**

```java
public ZeroNormException()
```

**Description**
The computations cannot continue because the Euclidean norm of the column is equal to zero.

**ZeroNormException**

```java
public ZeroNormException(string message)
```

**Description**
The computations cannot continue because the Euclidean norm of the column is equal to zero.

**Parameter**
- `message` – The error message that explains the reason for the exception.

**ZeroNormException**

```java
public ZeroNormException(string s, System.Exception exception)
```

**Description**
The computations cannot continue because the Euclidean norm of the column is equal to zero.

**Parameters**
- `s` – The error message that explains the reason for the exception.
- `exception` – The exception that is the cause of the current exception. If the innerException parameter is not a null reference, the current exception is raised in a catch block that handles the inner exception.

**ZeroNormException**

```java
```

**Description**
The computations cannot continue because the Euclidean norm of the column is equal to zero.

**Parameters**
- `info` – The object that holds the serialized object data.
- `context` – The contextual information about the source or destination.
Chapter 30: References

References

Abe

Abramowitz and Stegun

Afifi and Azen

Agresti, Wackerly, and Boyette
Agresti, Alan, Dennis Wackerly, and James M. Boyette (1979), Exact conditional tests for cross-classifications: Approximation of attained significance levels, Psychometrika, 44, 75-83.

Ahrens and Dieter

Akaike

Akaike et al

Akima

distributed data points, *ACM Transactions on Mathematical Software*, 4, 148-159.

**Anderberg**


**Anderson**


**Anderson and Bancroft**


**Ashcraft**


**Ashcraft et al.**


**Atkinson (1979)**


**Atkinson (1978)**


**Barrodale and Roberts**


**Bartlett, M. S.**


**Barnett**


**Barrett and Heal**


**Bays and Durham**


**Bendel and Mickey**


**Bentley and Sedgewick**


**Berry and Linoff**


**Best and Fisher**


**Bishop**


**Bishop et al**


**Bjorck and Golub**


**Blom**

Blom and Zegeling

Boisvert

Bosten and Battiste

Box and Jenkins

Box and Pierce

Boyette

Bradley

Breiman et al.

Brenan, Campbell, and Petzold

Brent

Breslow

Bridle

**Brigham**


**Brown**


**Brown and Benedetti**


**Burgoyne**


**Calvo**


**Carlson**


**Carlson and Notis**


**Carlson and Foley**


**Chen and Liu**


**Cheng**


**Chiang**


**Clarkson and Jenrich**

**Cohen and Taylor**


**Cooley and Tukey**


**Cooper**


**Conover**


**Cook and Weisberg**


**Courant and Hilbert**


**Cox**


**Cox and Oakes**


**Craven and Wahba**


**Crowe et al.**


**Davis and Rabinowitz**

Davis et al.

de Boor

Deming

Demmel et al.

Dennis and Schnabel

Dongarra et al.

Doornik

Draper and Smith

DuCroz et al.

Duff et al.

Duff and Reid
Elant-Johnson et al.

Elman

Eisenstat et al.

Emmett

Enright and Pryce

Farebrother and Berry

Fisher

Fishman and Moore

Forsythe

Franke

Furnival and Wilson

Garbow et al.
Garbow, B.S., J.M. Boyle, K.J. Dongarra, and C.B. Moler (1977), *Matrix Eigensystem Routines* -

**Gautschi**


**Gear**


**Gear and Petzold**


**Gentleman**


**George and Liu**


**Gill and Murray**


**Gill et al.**


**Giudici**


**Goldfarb and Idnani**


**Golub**


**Golub and Van Loan**


**Golub and Welsch**


**Gregory and Karney**


**Griffin and Redfish**


**Gross and Clark**


**Grosse**


**Guerra and Tapia**


**Hageman and Young**


**Hanson**


**Hardy**


**Harman**


**Hart et al.**


Higham, Nicholas J. (1988), FORTRAN Codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation, *ACM Transactions on Mathematical Software*, 14, 381-396.


Hocking, R.R. (1972), Criteria for selection of a subset regression: Which one should be used?, *Technometrics*, 14, 967-970.


Hull et al.

Hwang and Ding

Irvine et al.

Jackson et al.

Jacobs et al.

Jenkins

Jenkins and Traub


Jöhnk

Johnson and Kotz


1966 • IMSL C# Numerical Library

Jöreskog


Juran and Godfrey


Kachitvichyanukul


Kaiser


Kaiser and Caffrey


Kalbfleisch and Prentice


Kendall and Stuart


Kennedy and Gentle


Kernighan and Richtie


Kim and Jennrich


Kinnucan and Kuki


Kirk

References ● 1967

Kochanek and Bartels
Kochanek, Doris H. U., and Bartels, Richard H (1984), Interpolating Splines with Local Tension, Continuity, and Bias Control, ACM SIGGRAPH, vol. 18, no. 3, pp. 3341

Kohonen

Knuth

Krogh

Lachenbruch

Lawless

Lawrence et al

Lawson and Hanson

Lawson and Hanson

Learmonth and Lewis

Leavenworth

Lee
Lee, Elisa T. (1980), Statistical Methods for Survival Data Analysis, Lifetime Learning Publications,
Lehmann

Levenberg

Lentini and Pereyra

Lewis et al.

Li

Li

Liepman

Lippmann

Liu


References
Ljung and Box

Loh and Shih

Lyness and Giunta

Madsen and Sincovec

Maindonald

Mandic and Chambers

Manning and Schütze

Marquardt

Marsaglia

Martin and Wilkinson

Mayle

McCulloch and Pitts

1970 • IOMSL C# Numerical Library

Michelli


Michelli et al.


Microsoft Excel User Education Team


Miller


Milliken and Johnson


Moler and Stewart


Montgomery


Moré et al.


Müller


Murtagh


Murty

Neter and Wasserman

Neter et al.

NIST Engineering Statistics Handbook

Østerby and Zlatev

Owen


Pao

Parlett

Pennington and Berzins

Petro

Petzold

Piessens et al.

Poli and Jones
Powell

Pregibon

Quinlan
Quinlan, J. R. (1993), C4.5 Programs for Machine Learning, Morgan Kaufmann.

Ralston

Reed and Marks

Reinsch

Rice

Ripley

Rosenblatt

Rumelhart et al

References

1973


**Saad and Schultz**


**Sallas and Lionti**


**Savage**


**Schittkowski**


**Schmeiser**


**Schmeiser and Babu**


**Schmeiser and Kachitvichyanukul**


**Schmeiser and Lal**


---

**1974 ● IMSL C# Numerical Library**
Seidler and Carmichael

Shampine

Shampine and Gear

Shewart

Sincovec and Madsen

Singleton

Smirnov
Smirnov, N.V. (1939), Estimate of deviation between empirical distribution functions in two independent samples (in Russian), Bulletin of Moscow University, 2, 3–16.

Smith et al.

Smith

Smith

Spellucci, Peter
Spellucci, P. (1998), An SQP method for general nonlinear programs using only equality constrained subproblems, Math. Prog., 82, 413-448, Physica Verlag, Heidelberg, Germany


Spurrier and Isham

References

**Stewart**


**Stoer**


**Stoline**


**Strecok**


**Stroud and Secrest**


**Studenmund**


**Swingler**


**Taguchi**


**Temme**


**Tesauro**


**Tezuka**


**Thompson and Barnett**

Thompson, I.J. and A.R. Barnett (1987), Modified Bessel functions $I_n(z)$ and $K_n(z)$ of real order and complex argument, *Computer Physics Communication*, 47, 245-257.
Tukey

Velleman and Hoaglin

Verwer et al

Walker

Warner and Misra

Watkins

Weeks

Werbos

Western Electric

Williams and Zipser

Wilmott et al

References
Witten and Frank


Woodfield


Wu

Index

AbstractChartNode, 1294
Activation, 1664
AllConstraintsNotSatisfiedException, 1806
AllDeletedException, 1877
AllMissingException, 1878
AltSeriesAccuracyLossException, 1879
ANCOVA, 695
Annotation, 1342
ANOVA, 708
ComputeOption, 718
ANOVAFactorial, 719
ErrorCalculation, 729
ARAutoUnivariate, 841
ParamEstimation, 862
ARMA, 874
ParamEstimation, 895
ARMAEstimateMissing, 896
MissingValueEstimation, 905
ARMAMaxLikelihood, 906
ARMAOutlierIdentification, 919
ARSeasonalFit, 862
CenterMethod, 873
AutoARIMA, 940
InformationCriterion, 964
AutoCorrelation, 804
StdErr, 813
Axis, 1346
AxisID, 1353
AxisLabel, 1358
AxisLine, 1359
AxisR, 1365
AxisRLabel, 1367
AxisRLine, 1368
AxisRMajorTick, 1369
AxisTheta, 1370
AxisTitle, 1360
AxisUnit, 1360
AxisXY, 1348
Background, 1338
BadInitialGuessException, 1807
BadVarianceException, 1880
Bar, 1464
BarItem, 1473
BarSet, 1477
Bessel, 469
BinaryClassification, 1683
Bond, 1254
Frequency, 1288
BoundaryInconsistentException, 1808
BoundedLeastSquares, 400
IFunction, 408
IJacobian, 409
BoundedVariableLeastSquares, 409
BoundsInconsistentException, 1810
BoxPlot, 1420
Statistics, 1428
BsInterpolate, 172
BsLeastSquares, 174
BSpline, 168
Candlestick, 1458
CandlestickItem, 1460
CategoricalGenLinModel, 747
DistributionParameterModel, 770
CCChart, 1560
Cdf, 1101
Chart, 1332
ChartFunction, 1383
ChartNode, 1310
ChartSpline, 1384
ChartTitle, 1339
ChiSquaredTest, 783
Cholesky, 77
ClassificationVariableException, 1882
ClassificationVariableLimitException, 1883
ClassificationVariableValueException, 1884
ClusterHierarchical, 1009
  Linkage, 1018
  Transformation, 1019
ClusterKMeans, 993
ClusterNoPointsException, 1886
Colormap, 1517
Colormap.Fields, 1518
Complex, 475
ComplexFFT, 310
ComplexLU, 44
ComplexMatrix, 11
ComplexSparseCholesky, 88
  NumericFactor, 95
  NumericFactorization, 94
  SymbolicFactor, 95
ComplexSparseMatrix, 28
  SparseArray, 37
ComplexSuperLU, 63
  ColumnOrdering, 74
  PerformanceParameters, 75
  Scaling, 76
ConjugateGradient, 122
  IFunction, 129
  IPreconditioner, 130
ConstraintEvaluationException, 1811
ConstraintsInconsistentException, 1812
ConstraintsNotSatisfiedException, 1814
ConstrInconsistentException, 1887
ContingencyTable, 733
Contour, 1431
  Legend, 1444
  ContourLevel, 1445
  ControlLimit, 1527
  CorrectorConvergenceException, 1815
Covariances, 545
  MatrixType, 551
CovarianceSingular1Exception, 1889
CovarianceSingular2Exception, 1890
CovarianceSingularException, 1888
CrossCorrelation, 813
  StdErr, 826
CsAkima, 146
CsInterpolate, 152
  Condition, 154
CsPeriodic, 155
CsShape, 157
CsSmooth, 158
CsSmoothC2, 160
CsTCB, 148, 163
CuSum, 1569
CuSumStatus, 1572
CyclingIsOccurringException, 1890
CyclingOccurringException, 1816
Data, 1372
DayCountBasis, 1289
DeleteObservationsException, 1892
Dendrogram, 1484
DenseLP, 363
DidNotConvergeException, 1817, 1893
Difference, 965
DiffObsDeletedException, 1894
DiscriminantAnalysis, 1038
  Classification, 1061
  CovarianceMatrix, 1061
  Discrimination, 1060
  PriorProbabilities, 1062
Dissimilarities, 1002
  Method, 1007
  Scaling, 1009
Draw, 1393
DrawMap, 1414
DrawPick, 1405
Eigen, 132
EigenvalueException, 1896
EmpiricalQuantiles, 593
EmptyGroupException, 1897
EpochTrainer, 1680
EpsilonAlgorithm, 513
EqConstrInconsistentException, 1898
EqualityConstraintsException, 1819
ErrorBar, 1446
ErrorTestException, 1820
EWMA, 1566
FactorAnalysis, 1020
  MatrixType, 1037
  Model, 1037
MatrixType, 527
PrintMatrixFormat, 522
ColumnLabelType, 530
FormatType, 528
ParsePosition, 526
RowLabelType, 531
ProblemInfeasibleException, 1850
ProblemUnboundedException, 1851
ProblemVacuousException, 1852
ProportionalHazards, 1075
TieHandling, 1092
QPInfeasibleException, 1854
QPProblemUnboundedException, 1855
QR, 95
QuadraticProgramming, 385
Quadrature, 212
IFunction, 219
QuasiNewtonTrainer, 1668
IError, 1674
RadialBasis, 195
Gaussian, 207
HardyMultiquadric, 209
IFunction, 207
Random, 1182
BaseGenerator, 1206
RankDeficientException, 1937
RankException, 1936
Ranks, 584
Tie, 592
RChart, 1536
RegressorsForGLM, 625
DummyType, 633
ScaleFactorZeroException, 1938
ScaleFilter, 1742
ScalingMethod, 1751
SChart, 1546
SelectionRegression, 664
Criterion, 675
SummaryStatistics, 675
Sfun, 453
ShewhartControlChart, 1521
SignTest, 773
SingularException, 1856, 1940
SingularMatrixException, 1857
SingularPreconditionMatrixException, 1858
SingularTriangularMatrixException, 1941
SolutionNotFoundException, 1859
SomeConstraintsDiscardedException, 1860
Sort, 576
SparseCholesky, 81
NumericFactor, 88
NumericFactorization, 87
SymmetricFactor, 88
SparseMatrix, 16
SparseArray, 27
Spline, 143
Spline2D, 177
Spline2DInterpolate, 180
Spline2DLeastSquares, 189
SplineData, 1461
StepwiseRegression, 677
CoefficientTTestsValue, 687
Direction, 688
Summary, 534
SumOfWeightsNegException, 1942
SuperLU, 49
ColumnOrdering, 60
PerformanceParameters, 61
Scaling, 62
SVD, 99
SymEigen, 136
TableMultiWay, 608
TableBalanced, 614
TableUnbalanced, 615
TableOneWay, 596
TableTwoWay, 601
TcurrentTstopInconsistentException, 1862
TEqualsToutException, 1863
TerminationCriteriaNotSatisfiedException, 1865
Text, 1385
TimeIntervalTooSmallException, 1866
TimeSeriesClassFilter, 1764
TimeSeriesFilter, 1762
ToleranceTooSmallException, 1868
ToolTip, 1388
TooManyCallsException, 1944
TooManyFunctionEvaluationsException, 1945
TooManyIterationsException, 1869, 1946
TooManyIterationsReTryException, 1948
TooManyJacobianEvalException, 1949

Index
TooManyObsDeletedException, 1950
TooManyStepsException, 1871
TooMuchTimeException, 1872
Transform, 1362
TransformDate, 1363
Treemap, 1510

UChart, 1563
UnboundedBelowException, 1873
UnsupervisedNominalFilter, 1752
UnsupervisedOrdinalFilter, 1756
  TransformMethod, 1761
UserBasisRegression, 689

VarBoundsInconsistentException, 1875
VarsDeterminedException, 1951

Warning, 1795
WarningObject, 1797
WebChart, 1413
WilcoxonRankSum, 777
WorkingSetSingularException, 1876

XbarR, 1529
XbarS, 1539
XmR, 1550

ZeroNormException, 1952
ZeroPolynomial, 316
ZerosFunction, 320
  IFunction, 326
ZeroSystem, 327
  IFunction, 332
  IJacobian, 333