Visual Numerics®

IMSL™

C# Numerical Library

User's Guide

VERSION 4.0

IMSL™ C# Numerical Library V.4.0

User's Guide

Trusted for Over 30 Years

Visual Numerics Corporate Headquarters
2500 Wilcrest Drive
Houston, TX 77042

USA Contact Information

Toll Free:	800.222.4675
Houston, TX:	713.784.3131
Westminster, CO:	303.379.3040
Email:	info@vni.com
Web site:	www.vni.com

Visual Numerics has Offices Worldwide
USA • UK • France • Germany •Mexico
Japan • Korea • Taiwan
For contact information, please visit
www.vni.com/contact

C, C#, Java™, and Fortran
Application Development Tools

Contents

Chapter 1: Linear Systems

Types

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 x 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 x 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 x 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 x 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 x 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

$$A = \sum\nolimits_{i=1}^{q} s_{i,i} u_i v_i^T$$

then the approximate solution is given by the following:

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

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

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

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.

Matrix Class

Summary

Matrix manipulation functions.

`public class Imsl.Math.Matrix`

Methods

Add

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

Description

Add two rectangular matrixs, 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.

System.ArgumentException id is thrown when the matricies are not the same size

---

CheckSquareMatrix

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

Description

Check that the matrix is square.

Parameter

a – A double matrix.

System.ArgumentException id 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.

Multiply

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

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.

`System.ArgumentException` id 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`.

`System.ArgumentException` id 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)
```

Description

Multiply two rectangular matricies, a * b.

Parameters

`a` – A `double` rectangular matrix.

`b` – A `double` rectangular matrix.

Returns

The `double` matrix product of `a * b`.

`System.ArgumentException` id 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.

`System.ArgumentException` id is thrown when the matricies are not the same size

Transpose

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

Description

Return the transpose of an 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.

```
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}
        };

        //  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
   0  1  2  3
0  0  1  2  3
1  4  5  6  7
2  8  9  8  1
3  6  3  4  3

The 1 norm of the matrix is 20
```

ComplexMatrix Class

Summary

Complex matrix manipulation functions.

`public class Imsl.Math.ComplexMatrix`

Methods

Add

```
static public Imsl.Math.Complex[,] Add(Imsl.Math.Complex[,] a,
  Imsl.Math.Complex[,] b)
```

Description

Add two rectangular `Complex` arrays, a + b.

Parameters

`a` – A `Complex` rectangular array.

`b` – A `Complex` rectangular array.

Returns

The `Complex` matrix sum of the two arguments

`System.ArgumentException` id is thrown when the matricies 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.

`System.ArgumentException` id 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

```
static public Imsl.Math.Complex[] Multiply(Imsl.Math.Complex[] x,
  Imsl.Math.Complex[,] a)
```

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 * a.

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

---

Multiply

```
static public Imsl.Math.Complex[] Multiply(Imsl.Math.Complex[,] a,
  Imsl.Math.Complex[] x)
```

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 * x.

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

---

Multiply

```
static public Imsl.Math.Complex[,] Multiply(Imsl.Math.Complex[,] a,
  Imsl.Math.Complex[,] b)
```

Description

Multiply two `Complex` rectangular arrays, a * b.

Parameters

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

Returns

The `Complex` matrix product of `a` times `b`.

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

OneNorm

```
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

```
static public Imsl.Math.Complex[,] Subtract(Imsl.Math.Complex[,] a,
  Imsl.Math.Complex[,] b)
```

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.

`System.ArgumentException` id is thrown when the matricies are not the same size

Transpose

```
static public Imsl.Math.Complex[,] Transpose(Imsl.Math.Complex[,] a)
```

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;

public 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

```
      Matrix A
    0     1     2
0  1+3i  3+5i  7+9i
1  8+7i  9+5i  1+9i
2  2+9i  6+9i  7+3i
3  5+4i  8+4i  5+9i

       Transpose(A)
    0     1     2     3
0  1+3i  8+7i  2+9i  5+4i
1  3+5i  9+5i  6+9i  8+4i
2  7+9i  1+9i  7+3i  5+9i
```

LU Class

Summary

LU factorization of a matrix of type `double`.

`public class Imsl.Math.LU`

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.

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

Methods

Condition

`public double Condition(double[,] a)`

Description

Return an estimate of the reciprocal of the L1 condition number of a matrix.

Parameter

`a` – The `double` square matrix for which the reciprocal of the L1 condition number is desired.

Returns

A `double` value representing an estimate of the reciprocal of the L1 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.

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.

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.

`System.ArgumentException` id is thrown when the number of rows in the input matrix is not equal to the number of elements in x

`Imsl.Math.SingularMatrixException` id is thrown when the matrix is singular

SolveTranspose

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

Description

Return the solution x of the linear system transpose(A)x = 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.

Description

`LU` performs an LU factorization of a real general coefficient matrix. The `Condition` method 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 in a paper by Cline et al. (1979).

An estimated condition number greater than $1/\epsilon$ (where ϵ 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.

`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.

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.

```
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);
        Console.Out.WriteLine();

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

Output

```
  x
   0
0  3
1  2
2  1

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

condition number = 0.0151202749140893

determinant = -1
```

ComplexLU Class

Summary

LU factorization of a matrix of type `Complex`.

`public class Imsl.Math.ComplexLU`

Constructor

ComplexLU

`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.

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

Methods

Condition

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

Description

Return an estimate of the reciprocal of the L1 condition number.

Parameter

`a` – A `Complex` matrix.

Returns

A `double` scalar value representing the estimate of the reciprocal of the L1 condition number of the matrix `a`.

Determinant

`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.

Inverse

`public Imsl.Math.Complex[,] Inverse()`

Description

Compute the inverse of a matrix of type `Complex`.

Returns

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

Solve

```
public Imsl.Math.Complex[] Solve(Imsl.Math.Complex[] b)
```

Description

Return the solution x of the linear system ax = b using the LU factorization of 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

```
static public Imsl.Math.Complex[] Solve(Imsl.Math.Complex[,] a,
  Imsl.Math.Complex[] b)
```

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.

`System.ArgumentException` id is thrown when the number of rows in `a` is not equal to the length of `b`

`Imsl.Math.SingularMatrixException` id is thrown when the matrix is singular

SolveTranspose

```
public Imsl.Math.Complex[] SolveTranspose(Imsl.Math.Complex[] b)
```

Description

Return the solution x of the linear system transpose(A)x = b.

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.

Description

`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).

An estimated condition number greater than $1/\epsilon$ (where ϵ 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.

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 and the determinant and condition number of the matrix are obtained.

```
using System;
using Imsl.Math;

public 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);

        // Solve Ax = b
        Complex[] x = clu.Solve(b);
        Console.Out.WriteLine("The solution is:");
        Console.Out.WriteLine(" ");
        new PrintMatrix("x").Print(x);

        // Find the condition number of A.
        double condition = clu.Condition(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
0  3
1  2
2  1

The condition number = 0.0148867313915858

The determinant = -0.99999999999999978
```

Cholesky Class

Summary

Cholesky factorization of a matrix of type `double`.

`public class Imsl.Math.Cholesky`

Constructor

Cholesky

`public Cholesky(double[,] a)`

Description

Create the Cholesky factorization of a symmetric positive definite matrix of type `double`.

Parameter

`a` – A `double` square matrix to be factored.

`Imsl.Math.SingularMatrixException` id is thrown when the input matrix `a` is singular

`Imsl.Math.NotSPDException` id 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.

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

Parameter

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

`Imsl.Math.NotSPDException` id 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.

R is a lower triangular matrix and $A = RR^T$.

Returns

A `double` matrix which contains the R matrix that results from the Cholesky factorization.

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.

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

Parameter

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

Description

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 upper triangular matrix R and a permutation matrix P that satisfy $P^TAP = R^TR$, where P is represented by `ipvt`.

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

The Cholesky factorization of a matrix is $A = R^TR$, where R is an upper triangular matrix. Given this factorization, `Downdate` computes the factorization

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

`Downdate` 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 $U^T U = I$, the desired result

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

is obtained.

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

$$\alpha = \sqrt{1 - \|a\|_2^2}$$

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) * (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 θ_i.

The Givens rotations are then used to form

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

The matrix

$$\tilde{R}$$

is upper triangular and

$$\tilde{x} = x$$

because

$$x = \left(R^T 0\right) \begin{bmatrix} a \\ \alpha \end{bmatrix} = \left(R^T 0\right) U^T U \begin{bmatrix} a \\ \alpha \end{bmatrix} = \left(\tilde{R}^T \tilde{x}\right) \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \tilde{x}$$

.

Example: Cholesky Factorization

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

```
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
```

QR Class

Summary

QR Decomposition of a matrix.

`public class Imsl.Math.QR`

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

`public int 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.

`Imsl.Math.SingularMatrixException` id 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.

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

Description

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(number of rows of A, 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.

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.

```
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[]{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  0.990000000000019
1  2.00199999999999
2  3

                                          Q
            0                    1                    2                    3
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                  1                   2
0  -75.2595508889071  -10.6298800690547   -1.5944820103582
1    0                 -2.64681879196785  -1.15264689327632
2    0                  0                  0.359210604053549
3    0                  0                  0

rank = 3
```

SVD Class

Summary

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

`public class Imsl.Math.SVD`

Properties

Info

`public int Info {get; }`

Description

Returns the index of the first singular value for which the algorithm converged.

Rank

`public int Rank {get; }`

Description

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

Constructors

SVD

`public SVD(double[,] a, double tol)`

Description

Construct the singular value decomposition of a rectangular matrix with a given tolerance.

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.

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.

`Imsl.Math.DidNotConvergeException` id 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.

The tolerance used is 2.2204460492503e-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.

Parameter

`a` – A `double` matrix for which the singular value decomposition is to be computed.

Methods

GetS

```
public double[] GetS()
```

Description

Returns the singular values.

Returns

A `double` array containing the singular values of the matrix.

GetU

```
public double[,] GetU()
```

Description

Returns the left singular vectors.

Returns

A `double` matrix containing the left singular vectors.

GetV

```
public double[,] GetV()
```

Description

Returns the right singular vectors.

Returns

A `double` matrix containing the right singular vectors

Inverse

```
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.

Description

`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

n x p matrix A, there exists an n x n orthogonal matrix U and a p x p orthogonal matrix V such that

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

where $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_m)$, 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 σ_k that is larger than a tolerance η. If τ is the parameter `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 Σ as $U = (U_1, U_2)$, $V = (V_1, V_2)$ and $\Sigma_1 = \mathrm{diag}(\sigma_1, \ldots, \sigma_k)$ where the "1" matrices are k by k. The Moore-Penrose generalized inverse is $V_1 \Sigma_1^{-1} U_1^T$.

Example: Singular Value Decomposition of a Matrix

The singular value decomposition of a matrix is performed. The rank of the matrix is also computed.

```
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                    1                    2
0  -0.380475586320569   0.119670992640587    0.439082824383239
1  -0.403753713172442   0.345110837105607   -0.0565761852901658
2  -0.545120486248343   0.429264893493195    0.0513926928086694
3  -0.264784294004146  -0.0683195253271513  -0.883860867430429
4  -0.446310112301556  -0.816827623278282    0.141899675060401
5  -0.354628656614145  -0.102147399162125   -0.00431844397986985

           3                    4                    5
0  -0.565399585908374   0.0243115161463761  -0.57258686109915
1   0.214775576522681   0.80890058872827     0.11929741721493
2   0.432144162809737  -0.572327648171096    0.0403309248707933
3  -0.215253698182974  -0.0625209225900579  -0.30621669907105
4   0.321269584269887   0.0621337820958098  -0.0799352679998222
5  -0.545800221853259  -0.0987946265624981   0.745739576113111

          S
           0
0  11.4850179115597
1   3.2697512144125
2   2.65335616200783
3   2.08872967244092

                               V
           0                   1                   2
0  -0.444294128842354   0.555531257799947  -0.435378966673942
1  -0.558067238190387  -0.654298740112323   0.277456900458814
2  -0.32438610320628   -0.351360645592513  -0.732099533429598
```

```
3  -0.621238553843379   0.37393031038343    0.444401954223745

           3
0   0.55175438744187
1   0.428336065179864
2  -0.485128463324533
3  -0.526066236587424

rank = 4
```

Chapter 2: Eigensystem Analysis

Types

Usage Notes

An ordinary linear eigensystem problem is represented by the equation $Ax = \lambda x$ where A denotes an n x n matrix. The value λ 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 ε is the machine precision. By a theorem due to Bauer and Fike (see Golub and Van Loan 1989, p. 342),

$$\min\left|\tilde{\lambda} - \lambda\right| \leq \kappa\,(X)\,\|E\|_2 \quad \text{for all}\,\lambda\,\text{in}\,\sigma\,(A)$$

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 $\kappa(X) = \|X\|_2 \left\|X^{-1}\right\|_2$. 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

$$\tilde{\lambda}_j$$

and eigenvectors

$$\tilde{x}_j$$

can be checked by computing their performance index τ. The performance index is defined to be

$$\tau = \max_{1 \le j \le n} \frac{\left\|A\tilde{x}_j - \tilde{\lambda}_j \tilde{x}_j\right\|_2}{n\varepsilon \|A\|_2 \|\tilde{x}_j\|_2}$$

where ε is again the machine precision.

The performance index τ is related to the error analysis because

$$\|E\tilde{x}_j\|_2 = \left\|A\tilde{x}_j - \tilde{\lambda}_j \tilde{x}_j\right\|_2$$

where E is the "nearby" matrix discussed above.

While the exact value of τ is precision and data dependent, the performance of an eigensystem analysis function is defined as excellent if $\tau < 1$, good if $1 \le \tau \le 100$, and poor if $\tau > 100$. This is an arbitrary definition, but large values of τ 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 τ 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 λ_i is

$$\kappa_j = \left\|e_j^T X^{-1}\right\|,$$

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 $\kappa_j \varepsilon \|A\|$. To compute an approximate bound for the relative accuracy of an eigenvalue, divide this bound by $|\lambda_j|$.

Eigen Class

Summary

Collection of Eigen System functions.

`public class Imsl.Math.Eigen`

Constructors

Eigen

`public Eigen(double[,] a)`

Description

Constructs the eigenvalues and the eigenvectors of a real square matrix.

Parameter

`a` – A `double` square matrix whose eigensystem is to be constructed.

`Imsl.Math.DidNotConvergeException` id is thrown when the algorithm fails to converge on the eigenvalues of the matrix

Eigen

`public Eigen(double[,] a, bool computeVectors)`

Description

Constructs the eigenvalues and (optionally) the eigenvectors of a real square matrix.

Parameters

`a` – A `double` square matrix whose eigensystem is to be constructed.

`computeVectors` – A `bool` value of `true` if the eigenvectors are to be computed.

`Imsl.Math.DidNotConvergeException` id is thrown when the algorithm fails to converge on the eigenvalues of the matrix

Methods

GetValues

`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

```
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

```
public double PerformanceIndex(double[,] a)
```

Description

Returns the performance index of a real eigensystem.

A performance index less than 1 is considered excellent, 1 to 100 is good, while greater than 100 is considered poor.

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.

Description

`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, τ, is highly machine dependent, the performance of `Eigen` is considered excellent if $\tau < 1$, good if $1 \le \tau \le 100$, 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).

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(a);
        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

                Eigenvectors
                      0
0     0.316227766016838-0.316227766016838i
1     0.632455532033676
2  1.66533453693773E-16-0.632455532033676i

                      1
0     0.316227766016838+0.316227766016838i
1     0.632455532033676
2  1.66533453693773E-16+0.632455532033676i

           2
0  0.408248290463863
1  0.816496580927725
2  0.408248290463864
```

SymEigen Class

Summary

Computes the eigenvalues and eigenvectors of a real symmetric matrix.

`public class Imsl.Math.SymEigen`

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 `boolean`, `true` if the eigenvectors are to be computed.

Methods

GetValues

`public double[] GetValues()`

Description

Returns the eigenvalues.

If the algorithm fails to converge on an eigenvalue, that eigenvalue is set to `NaN`.

Returns

A `double` array containing the eigenvalues in descending order.

GetVectors

`public double[,] GetVectors()`

Description

Return the eigenvectors of a symmetric matrix of type `double`.

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.

Returns

A `double` array containing the eigenvectors.

PerformanceIndex

`public double PerformanceIndex(double[,] a)`

Description

Returns the performance index of a real symmetric eigensystem.

A performance index less than 1 is considered excellent, 1 to 100 is good, while greater than 100 is considered poor.

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.

Description

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 ε be the machine precision. The performance index, τ, is defined to be

$$\tau = \max_{1 \le j \le M} \frac{\|Ax_j - \lambda_j x_j\|_1}{10N\varepsilon \|A\|_1 \|x_j\|_1}$$

While the exact value of τ is highly machine dependent, the performance of `SymEigen` is considered excellent if $\tau < 1$, good if $1 \le 100$, 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).

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
0   3
1  -3.62597321469472E-16
2  -2.22044604925031E-16

                         Eigenvectors
           0                   1                   2
0  0.577350269189626   0.816496580927726   0
1  0.577350269189626  -0.408248290463863  -0.707106781186547
2  0.577350269189626  -0.408248290463863   0.707106781186548
```

Chapter 3: Interpolation and Approximation

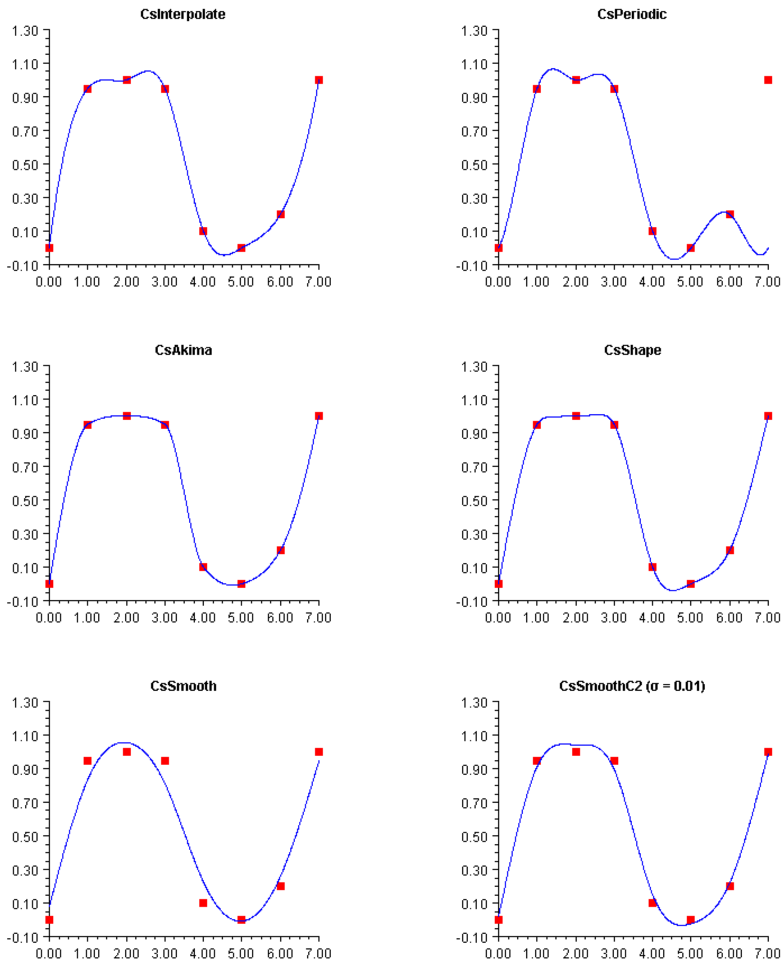
Types

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 appoximating 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 dervived 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 derviatives 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.

Spline Class

Summary

`Spline` represents and evaluates univariate piecewise polynomial splines.

`public class Imsl.Math.Spline`

Constructor

Spline

`Spline()`

Description

Initializes a new instance of the Imsl.Math.Spline (p. 43) class.

Methods

Derivative

`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.

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

`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.

Eval

`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.

Eval

`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.

GetBreakpoints

`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.

Description

A univariate piecewise polynomial (function) *p(x)* is specified by giving its breakpoint sequence `breakPoint[]`$= \xi \in \mathbf{R}^n$, the order k (degree k-1) of its polynomial pieces,and the $k \times (n-1)$ matrix `coef`$=c$ of its local polynomial coefficients. In terms of this information, the piecewise polynomial (ppoly) function is given by

$$p(x) = \sum_{j=1}^{k} c_{ji} \frac{(x-\xi_i)^{j-1}}{(j-1)!} \quad \text{for } \xi_i \le x \le \xi_{i+1}$$

The breakpoint sequence ξ 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.

CsAkima Class

Summary

Extension of the Spline class to handle the Akima cubic spline.

```
public class Imsl.Math.CsAkima :  Spline
```

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.

`System.ArgumentException` id is thrown if the arrays `xData` and `yData` do not have the same length

Description

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 ξ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$\|f - s\|_{[\xi_0, \xi_{n-1}]} \leq C \left\| f^{(2)} \right\|_{[\xi_0, \xi_{n-1}} |\xi|^2$$

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.)

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.

```
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 .25 is " + csv);
    }
}
```

Output

```
The computed cubic spline value at point .25 is -0.478185519991867
```

CsInterpolate Class

Summary

Extension of the Spline class to interpolate data points.

`public class Imsl.Math.CsInterpolate : Spline`

Constructors

CsInterpolate

`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

```
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.

`typeRight` – A `CsInterpolate.Condition` denoting the type of condition at the right endpoint. This can be `NotAKnot`, `FirstDerivative` or `SecondDerivative`.

`valueRight` – A `double` value at the right endpoint.

Description

`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 ξ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$|f - s|_{[\xi_0,\xi_n]} \le C \left\| f^{(4)} \right\|_{[\xi_0,\xi_n]} |\xi|^4$$

where

$$|\xi| := \max_{i=0,\ldots,n-1} |\xi_{i+1} - \xi_i|$$

For more details, see de Boor (1978, pages 55-56).

Example: The cubic spline interpolant

A cubic spline interpolant to a function is computed. The value of the spline at point 0.25 is printed.

```
using System;
using Imsl.Math;

public class CsInterpolateEx1
{
    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] = System.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

Summary

Denotes the type of condition at an endpoint.

`public enumeration Imsl.Math.CsInterpolate.Condition`

Fields

FirstDerivative
`public Imsl.Math.CsInterpolate.Condition FirstDerivative`

Description

Satisfies the endpoint condition of the first derivative at the right and left points.

`NotAKnot`

`public Imsl.Math.CsInterpolate.Condition NotAKnot`

Description

Satisfies the "not-a-knot" condition.

`SecondDerivative`

`public Imsl.Math.CsInterpolate.Condition SecondDerivative`

Description

Satisfies the endpoint condition of the second derivative at the right and left points.

CsPeriodic Class

Summary

Extension of the Spline class to interpolate data points with periodic boundary conditions.

`public class Imsl.Math.CsPeriodic :  Spline`

Constructor

CsPeriodic

`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.

Description

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 ξ 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).

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.

```
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

Summary

Extension of the Spline class to interpolate data points consistent with the concavity of the data.

`public class Imsl.Math.CsShape : Spline`

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.

`Imsl.Math.TooManyIterationsException` id is thrown if the iteration did not converge.

`Imsl.Math.SingularMatrixException` id is thrown if matrix is singular.

Description

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.

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.

```
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

Summary

Extension of the Spline class to construct a smooth cubic spline from noisy data points.

`public class Imsl.Math.CsSmooth :  Spline`

Constructors

CsSmooth

```
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

```
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 havethe same length as `xData`.

Description

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

$$\sum_{i=0}^{n-1} \left|(S(x_i) - f_i)w_i\right|^2 \le \sigma$$

where σ 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 σ. `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 σ so that the smoothing spline (S_σ) 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).

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
        Imsl.Stat.Random rn = new Imsl.Stat.Random(1234579);
        rn.Multiplier = 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 " +
                              "point .3010 is " + csv);
    }
}
```

Output

```
The computed cubic spline value at point .3010 is 0.0101201298963992
```

CsSmoothC2 Class

Summary

Extension of the Spline class used to construct a spline for noisy data points using an alternate method.

`public class Imsl.Math.CsSmoothC2 :  Spline`

Constructors

CsSmoothC2

`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

`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.

Description

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_σ is the unique C^2 function that minimizes

$$\int_a^b s''_\sigma(x)^2\,dx$$

subject to the constraint

$$\sum_{i=0}^{n-1} |s_\sigma(x_i) - f_i|^2 \le \sigma$$

.

Recommended values for σ depend on the weights, w. If an estimate for the standard deviation of the error in the y-values is availiable, then w_i should be set to this value and the smoothing parameter should be choosen in the confidence interval corresponding to the left side of the above inequality. That is,

$$n - \sqrt{2n} \le \sigma \le n + \sqrt{2n}$$

CsSmoothC2 is based on an algorithm of Reinsch (1967). This algorithm is also discussed in de Boor (1978, pages 235-243).

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.

```
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
        Imsl.Stat.Random rn = new Imsl.Stat.Random(1234579);
        rn.Multiplier = 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
```

BSpline Class

Summary

Spline represents and evaluates univariate B-splines.

`public class Imsl.Math.BSpline`

Constructor

BSpline

`BSpline()`

Description

Initializes a new instance of the Imsl.Math.BSpline (p. 58) 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.

If `ideriv` is zero, the function value is returned. If one, the first derivative is returned, etc.

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.

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.

If `ideriv` is zero, the function value is returned. If one, the first derivative is returned, etc.

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`.

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.

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`.

Description

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 $\mathbf{t} \in \mathbf{R}^M$. 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 ξ_i, then the number ξ_i should occur k - j - 1 times in the knot sequence. Assuming that ξ_1 and ξ_n are the endpoints of the interval of interest, choose the first k knots equal to ξ_1 and the last k knots equal to ξ_n. This can be done because the B-splines are defined to be right continuous near ξ_1 and left continuous near ξ_n.

When the above construction is completed, a knot sequence $\mathbf{t}$ of length M is generated, and there are $m{:}=M\text{-}k$ B-splines of order k, for example $B_0, ..., B_{m-1}$, spanning the ppoly functions on the interval with the indicated smoothness. That is, each ppoly function in this class has a unique representation $p = a_0 B_0 + a_1 B_1 + ... + a_{m-1} B_{m-1}$ 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 $[\mathbf{t}_i, \mathbf{t}_{i+k}]$. 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.

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 "
                              + ".23 is " + bsv);
    }
}
```

Output

```
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.

```
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
```

BsInterpolate Class

Summary

Extension of the BSpline class to interpolate data points.

`public class Imsl.Math.BsInterpolate :  BSpline`

Constructors

BsInterpolate

`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

`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

```
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.

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.

Description

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

$x_i < x_{i+1}$ for $i = 0, \ldots, n-2$

$\mathbf{t}_i < \mathbf{t}_{i+k}$ for $i = 0, \ldots, n-1$

$\mathbf{t}_i < \mathbf{t}_{i+1}$ 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 $\mathbf{t}_{k-1} \leq x_i \leq \mathbf{t}_n$ 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, $B_{j-k+1}, ..., B_j$ where j satisfies $\mathbf{t}_j \leq x_i < \mathbf{t}_{j+1}$ be well-defined (that is, $j - k + 1 \geq 0$).

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 "
                              + ".23 is " + bsv);
    }
}
```

Output

```
The computed B-spline value at point .23 is -0.303418399276769
```

BsLeastSquares Class

Summary

Extension of the BSpline class to compute a least squares spline approximation to data points.

```
public class Imsl.Math.BsLeastSquares :  BSpline
```

Constructors

BsLeastSquares

```
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

```
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

```
public 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`a and `weight` must have the same length.

`knot` – A `double` array containing the knot sequence for the spline.

Description

Let's make the identifications

$n =$ xData.length

$x =$ xData

$f =$ yData

$m =$ nCoef

$k =$ 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 $\mathbf{t}$ 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 $B_j, j = 1, ..., m$ are a (B-spline) basis for the spline subspace.

This algorithm is based on the routine L2APPR by deBoor (1978, p. 255).

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.

```
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
```

RadialBasis Class

Summary

Computes a least-squares fit to scattered data.

`public class Imsl.Math.RadialBasis`

Properties

ANOVA

`public Imsl.Stat.ANOVA ANOVA {get; }`

Description

Returns the `ANOVA` statistics from the linear regression.

See Also: Imsl.Stat.LinearRegression (p. 326), Imsl.Stat.ANOVA (p. 383)

RadialFunction

`public Imsl.Math.RadialBasis.IFunction RadialFunction {get; set; }`

Description

The 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 location of the data point at which the approximation is to be computed.

Returns

The value of the radial basis approximation at x.

Eval

`public double[] Eval(double[,] x)`

Description

Returns the value of the radial basis approximation at a point.

Parameter

`x` – A `double[,]`, the point at which the radial basis is to be evaluated.

Returns

A `double[]` 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 location 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 location 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 sepecified weight.

Parameters

`x` – A `double` array containing the location 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 *nPoints* by *nDim* containing the location of the data points.

`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 *nPoints* by *nDim* containing the location of the data points.

`f` – A `double` array containing the function values at the data points.

`w` – A `double` array containing the weights associated with the data points.

Description

RadialBasis computes a least-squares fit to scattered data in $\mathbf{R}^d$, where d is the dimension. More precisely, we are given data points

$$x_0, \ldots, x_{n-1} \in \mathbf{R}^d$$

and function values

$$f_0, \ldots, f_{n-1} \in \mathbf{R}^1$$

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

$$\sum_{i=0}^{n-1} w_i \left(F(x_i) - f_i\right)^2$$

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 $\mathbf{R}^1$. 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^{-ax^2}.

By default, the centers are points in a Faure sequence, scaled to cover the box containing the data.

Example: Radial Basis Function Approximation

The function

$$e^{-\|\vec{x}\|^2/d}$$

where d is the dimension, is evaluated at a set of randomly choosen 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 choosen 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.

```
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 choosen points
        int nData = 200;
        double[,] xData = new double[nData,nDim];
        double[] fData = new double[nData];
        Imsl.Stat.Random rand = new Imsl.Stat.Random(234567);
        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 = System.Math.Abs(fcn(x) - rb.Eval(x));
            aveError += error;
            maxError = System.Math.Max(error, maxError);
            double f = fcn(x);
        }
        aveError /= nTest;

        Console.Out.WriteLine("average error is " + aveError);
        Console.Out.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];
        }
        sum /= x.Length;
        return System.Math.Exp(-sum);
    }
}
```

Output

```
average error is 0.041978979550254
maximum error is 0.171666811944546
```

RadialBasis.IFunction Interface

Summary

Public interface for the user supplied function to the `RadialBasis` object.

`public interface Imsl.Math.RadialBasis.IFunction`

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.

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

Summary

The Gaussian basis function, e^{-ax^2}.

`public class Imsl.Math.RadialBasis.Gaussian : Imsl.Math.RadialBasis.IFunction`

Constructor

Gaussian

`public Gaussian(double a)`

Description

The Gaussian basis function, e^{-ax^2}.

Parameter

`a` – A `double`, the value of the function at x

Methods

F

`Final 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

`Final public double G(double x)`

Description

The derivative of the 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.

RadialBasis.HardyMultiquadric Class

Summary

The Hardy multiquadric basis function, $\sqrt{r^2 + \delta^2}$.

```
public class Imsl.Math.RadialBasis.HardyMultiquadric :
Imsl.Math.RadialBasis.IFunction
```

Constructor

HardyMultiquadric

```
public HardyMultiquadric(double delta)
```

Description

Creates a Hardy multiquadric basis function.

Parameter

`delta` – The parameter in the function definition.

Methods

F

```
Final 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

`Final public double G(double x)`

Description

The derivative of the 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.

Chapter 4: Quadrature

Types

Usage Notes

Univariate Quadrature

Class Quadrature computes approximations to integrals of the form

$$\int_c^b f(x)dx$$

Quadrature computes an estimated answer R. An optional value `ErrorEstimate` $= E$ estimates the error. These numbers are related as follows:

$$\left|\int_a^b f(x)\,dx - R\right| \le E \le \max\left\{\epsilon, \rho\left|\int_a^b f(x)\,dx\right|\right\}$$

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 a 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.

$$\int_{a_1}^{b_1} \ldots \int_{a_n}^{b_n} f\left(x_1,\ \ldots,\ x_n\right) dx_n \ldots dx_1$$

Quadrature Class

Summary

`Quadrature` is a general-purpose integrator that uses a globally adaptive scheme in order to reduce the absolute error.

```
public class Imsl.Math.Quadrature
```

Properties

AbsoluteError

```
public double AbsoluteError {get; set; }
```

Description

The absolute error tolerance.

ErrorEstimate

```
public double ErrorEstimate {get; }
```

Description

Returns an estimate of the relative error in the computed result.

ErrorStatus

```
public int ErrorStatus {get; }
```

Description

Returns the non-fatal error status.

Extrapolation

```
public bool Extrapolation {get; set; }
```

Description

If `true`, the epsilon-algorithm for extrapolation is enabled.

The default is `false` (extrapolation is not used).

MaxSubintervals

```
public int MaxSubintervals {get; set; }
```

Description

The maximum number of subintervals allowed.

The default value is 500.

RelativeError

```
public double RelativeError {get; set; }
```

Description

The relative error tolerance.

Rule

```
public int Rule {get; set; }
```

Description

The Gauss-Kronrod rule.

The default is 3.

Rule	Data points used
1	7 - 15
2	10 - 21
3	15 - 31
4	20 - 41
5	25 - 51
6	30 - 61

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.

Description

`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. This class is based on the subroutine `QAG` by Piessens et al. (1983).

Example 1: Integral $\int_1^3 e^{2x}\,dx$

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 $\int_0^\infty e^{-x}\,dx$

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);
    }
}
```

Output

```
result = 0.999999999999999
expect = 1
```

Example 3: Integral of the entire real line

The integral $\int_{-\infty}^{\infty} \frac{x}{4e^x+9e^{-x}}\,dx$ is computed and compared to its expected value. This integral is evaluated in Gradshteyn and Ryzhik (equation 3.417.1).

```
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

Gradshteyn, I. S. and I. M. Ryzhik (1965), *Table of Integrals, Series, and Products*, Academic Press, New York.

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.

```
using System;
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 = -3.05614388902526E-05
expect = -3.05614388888252E-05
relative error = -4.67047941622356E-11
relative error estimate = 1.04883755414239E-08
```

Quadrature.IFunction Interface

Summary

Interface defining function for the `Quadrature` class.

```
public interface Imsl.Math.Quadrature.IFunction
```

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

Summary

HyperRectangleQuadrature integrates a function over a hypercube.

`public class Imsl.Math.HyperRectangleQuadrature`

Properties

AbsoluteError

`public double AbsoluteError {get; set; }`

Description

Sets the absolute error tolerance.

ErrorEstimate

`public double ErrorEstimate {get; }`

Description

Returns an estimate of the relative error in the computed result.

RelativeError

`public double RelativeError {get; set; }`

Description

Sets 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.

Eval

```
public double Eval(Imsl.Math.HyperRectangleQuadrature.IFunction f, double[]
  a, double[] b)
```

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.

Description

This class is used to evaluate integrals of the form:

$$\int_{a_{n-1}}^{b_{n-1}} \cdots \int_{a_0}^{b_0} f(x_0, \ldots, x_{n-1})\, dx_0 \ldots dx_{n-1}$$

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. 688).

Example: HyperRectangle Quadrature

This example evaluates the following multidimensional integral, with n=10.

$$\int_{a_{n-1}}^{b_{n-1}} \cdots \int_{a_0}^{b_0} \left[\sum_{i=0}^{n} (-1)^i \prod_{j=0}^{i} x_j \right] dx_0 \ldots dx_{n-1} = \frac{1}{3}\left[1 - \left(-\frac{1}{2}\right)^n\right]$$

```
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

Summary

Interface for the HyperRectangleQuadrature function.

`public interface Imsl.Math.HyperRectangleQuadrature.IFunction`

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

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

$$\frac{dy_i}{dt} = y_i' = f_i\left(t,\, y_1,\, \ldots, y_N\right) \qquad i = 1,\, \ldots,\, N$$

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,

$$y\left(t\right)\left[y_1\left(t\right), \ldots, y_N\left(t\right)\right]^T, \left[f_1\left(t, y\right), \ldots, f_N\left(t, y\right)\right]^T$$

so that the problem becomes $y' = f\left(t, y\right)$ with initial values $y(t_0)$.

The system

$$\frac{dy}{dt} = y' = f\left(t, y\right)$$

is said to be *stiff* if some of the eigenvalues of the Jacobian matrix

$$\{\partial y_i'/\partial y_j\}$$

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.

OdeRungeKutta Class

Summary

Solves an initial-value problem for ordinary differential equations using the Runge-Kutta-Verner fifth-order and sixth-order method.

```
public class Imsl.Math.OdeRungeKutta
```

Properties

InitialStepsize

```
public double InitialStepsize {get; set; }
```

Description

The initial internal step size.

MaximumStepsize

```
public double MaximumStepsize {get; set; }
```

Description

The maximum internal step size.

MaxSteps

```
public int MaxSteps {get; set; }
```

Description

The maximum number of internal steps allowed.

MinimumStepsize

`public double MinimumStepsize {get; set; }`

Description

The minimum internal step size.

Scale

`public double Scale {get; set; }`

Description

The scaling factor.

Tolerance

`public double Tolerance {get; set; }`

Description

The error tolerance.

Constructor

OdeRungeKutta

`public OdeRungeKutta(Imsl.Math.OdeRungeKutta.IFunction f)`

Description

Constructs an ODE solver to solve the initial value problem dy/dx = f(x,y).

Parameter

`f` – Implementation of interface `IFunction` that defines the right-hand side function f(x,y).

Methods

Solve

`public void Solve(double x, double xEnd, double[] y)`

Description

Integrates the ODE system from x to xEnd.

On all but the first call to solve, the value of x must equal the value of xEnd for the previous call.

Parameters

`x` – A `double` specifying the independent variable.

`xEnd` – A `double` specifying the value of x at which the solution is desired.

`y` –
On input, `double` array containing the initial values.
On output, `double` array containing the approximate solution.

`Imsl.Math.MaxNumberStepsAllowedException` id 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` id is thrown if the computation does not converge on some step

---

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.

Description

Class `OdeRungeKutta` finds an approximation to the solution of a system of first-order differential equations of the form $y_0 = f(t, y)$ with given initial data. The routine attempts to keep the global error proportional to a user-specified tolerance. This routine 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.

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.

```
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);
        }
        Console.Out.WriteLine("Result = {" + y[0] + "," + y[1] + "}");
    }
}
```

Output

```
Result = {3.14434167651608,0.348826598519701}
```

OdeRungeKutta.IFunction Interface

Summary

Interface for user supplied function to `OdeRungeKutta` object.

`public interface Imsl.Math.OdeRungeKutta.IFunction`

Method

F

`abstract public void F(double x, double[] y, double[] yprime)`

Description

User supplied function to `OdeRungeKutta` object. On return, `yprime` contains the function value at the given point.

Parameters

`x` – 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, on return, contains the value of the function at (x,y).

Chapter 6: Transforms

Types

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

$$\hat{f}\left(\omega\right) = \left(\Im f\right)\left(\omega\right) = \int_{-\infty}^{\infty} f\left(t\right) e^{-2\pi i \omega t} dt$$

We begin by making the following approximation:

$$\hat{f}(\omega) \approx \int_{-T/2}^{T/2} f(t) e^{-2\pi i \omega t} dt$$

$$= \int_0^T f(t - T/2) e^{-2\pi i \omega (t - T/2)} dt$$

$$= e^{\pi i \omega T} \int_0^T f(t - T/2) e^{-2\pi i \omega t} dt$$

If we approximate the last integral using the rectangle rule with spacing $h = T/n$, we have

$$\hat{f}(\omega) \approx e^{\pi i \omega T} h \sum_{k=0}^{n-1} e^{-2\pi i \omega k h} f(kh - T/2)$$

Finally, setting $\omega = j/T$ for $j = 0, \ldots, n-1$ yields

$$\hat{f}(j/T) \approx e^{\pi i j} h \sum_{k=0}^{n-1} e^{-2\pi i j k/n} f(kh - T/2) = (-1)^j \sum_{k=0}^{n-1} e^{-2\pi i j k/n} f_k^h$$

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

Summary

FFT functions.

```
public class Imsl.Math.FFT
```

Constructor

FFT

```
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

`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

`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.

Description

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

$$q_{2m-1} = \sum_{k=0}^{n-1} p_k \cos \frac{2\pi km}{n} \quad m = 1,\ \ldots,\ n/2$$

$$q_{2m-2} = -\sum_{k=0}^{n-1} p_k \sin \frac{2\pi km}{n} \quad m = 1,\ \ldots,\ n/2-1$$

$$q_0 = \sum_{k=0}^{n-1} p_k$$

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 δ seconds starting at time t_0. That is, we have

$$p_i := f\left(t_0 + i\Delta\right)\ i = 0,\, 1, \ldots,\, n-1$$

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 $f\left(t_0\right) = f\left(t_0 + n\Delta\right)$. Hence, the period of the function is assumed to be $T = n\Delta$. We can invert the above transform for p as follows:

$$p_m = \frac{1}{n}\left[q_0 + 2\sum_{k=0}^{(n-3)/2} q_{2k+1}\cos\frac{2\pi(k+1)m}{n} - 2\sum_{k=0}^{(n-3)/2} q_{2k+2}\sin\frac{2\pi(k+1)m}{n}\right]$$

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

$$g\left(t\right) = \frac{1}{n}\left[q_0 + 2\sum_{k=0}^{(n-3)/2} q_{2k+1}\cos\frac{2\pi(k+1)\left(t-t_0\right)}{n\Delta} - 2\sum_{k=0}^{(n-3)/2} q_{2k+2}\sin\frac{2\pi(k+1)\left(t-t_0\right)}{n\Delta}\right]$$

$$= \frac{1}{n}\left[q_0 + 2\sum_{k=0}^{(n-3)/2} q_{2k+1}\cos\frac{2\pi(k+1)\left(t-t_0\right)}{T} - 2\sum_{k=0}^{(n-3)/2} q_{2k+2}\sin\frac{2\pi(k+1)\left(t-t_0\right)}{T}\right]$$

then we have

$$f\left(\mathrm{t}_0 + (i-1)\,\Delta\right) = g\left(\mathrm{t}_0 + (i-1)\right)\Delta$$

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

$$\frac{k}{T} = \frac{k}{n\Delta} \qquad k = 0,\ 1,\ \ldots,\ \frac{n-1}{2}$$

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.

Example: Fast Fourier Transform

The Fourier coefficients of a periodic sequence are computed. The coefficients are then used to reproduce the periodic sequence.

```
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);
        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
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

Summary

Complex FFT.

`public class Imsl.Math.ComplexFFT`

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

`public Imsl.Math.Complex[] Backward(Imsl.Math.Complex[] coef)`

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

`public Imsl.Math.Complex[] Forward(Imsl.Math.Complex[] seq)`

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.

Description

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

$$c_m = \sum_{n=0}^{N-1} x_n e^{-2\pi i n m/N}$$

Furthermore, a vector of Euclidean norm S is mapped into a vector of norm

$$\sqrt{N}S$$

Finally, note that we can invert the Fourier transform as follows:

$$x_n = \frac{1}{N} \sum_{j=0}^{N-1} c_m e^{2\pi i n j/N}$$

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

$$s_m = \sum_{n=0}^{N} c_n e^{2\pi i n m/N}$$

Furthermore, a vector of Euclidean norm S is mapped into a vector of norm

$$\sqrt{N}S$$

Finally, note that we can invert the inverse Fourier transform as follows:

$$c_n = \frac{1}{N} \sum_{m=0}^{N-1} s_m e^{-2\pi i n m/N}$$

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.

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+8i
1  2+7i
2  3+6i
```

```
3  4+5i
4  5+4i
5  6+3i
6  7+2i
7  8+1i

                  y
                   0
0                 36+36i
1   5.65685424949238+13.6568542494924i
2                   +8i
3  -2.34314575050762+5.65685424949238i
4                 -4+4i
5  -5.65685424949238+2.34314575050762i
6                 -8
7  -13.6568542494924-5.65685424949238i

    z
    0
0  1+8i
1  2+7i
2  3+6i
3  4+5i
4  5+4i
5  6+3i
6  7+2i
7  8+1i
```

Chapter 7: Nonlinear Equations

Types

Usage Notes

Zeros of a Polynomial

A polynomial function of degree n can be expressed as follows:

$$p(z) = a_n z^n n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0$$

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 `ZeroFunction` 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:

$$f_i(x) = 0, \text{for} \quad i = 1, 2, \ldots, n$$

where $x \in \mathbf{R}^n$, and $f_i : \mathbf{R^n} \rightarrow \mathbf{R}$. 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

Summary

The ZeroPolynomial class computes the zeros of a polynomial with complex coefficients, Aberth's method.

```
public class Imsl.Math.ZeroPolynomial
```

Property

MaximumIterations

```
public int MaximumIterations {get; set; }
```

Description

The maximum number of iterations allowed.

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.

$$p(x) = \mathrm{coef}[n] \times x^n + \mathrm{coef}[n-1] \times x^{n-1} + \ldots + \mathrm{coef}[0]$$

Parameter

`coef` – A `double` array containing the polynomial coefficients.

Returns

A `Complex` array containing the roots of the polynomial.

`Imsl.Math.DidNotConvergeException` id is thrown if the iteration did not converge.

ComputeRoots

`public Imsl.Math.Complex[] ComputeRoots(Imsl.Math.Complex[] coef)`

Description

Computes the roots of the polynomial with Complex coefficients.

$$p(x) = \mathrm{coef}[n] \times x^n + \mathrm{coef}[n-1] \times x^{n-1} + \ldots + \mathrm{coef}[0]$$

Parameter

`coef` – A `Complex` array containing the polynomial coefficients.

Returns

A `Complex` array containing the roots of the polynomial.

`Imsl.Math.DidNotConvergeException` id 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.

`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.

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.

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.

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.

Parameter

`index` – An `int` representing the (0-based) index of the root whose error status is to be returned.

Returns

A `boolean` representing the error status on the index-th root.

Description

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.

THIS SOFTWARE IS BEING PROVIDED ”AS IS”, WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN NO EVENT, NEITHER THE AUTHORS, NOR THE PUBLISHER, NOR ANY MEMBER OF THE EDITORIAL BOARD OF THE JOURNAL ”NUMERICAL ALGORITHMS”, NOR ITS EDITOR-IN-CHIEF, BE LIABLE FOR ANY ERROR IN THE SOFTWARE, ANY MISUSE OF IT OR ANY DAMAGE ARISING OUT OF ITS USE. THE ENTIRE RISK OF USING THE SOFTWARE LIES WITH THE PARTY DOING SO. ANY USE OF THE SOFTWARE CONSTITUTES ACCEPTANCE OF THE TERMS OF THE ABOVE STATEMENT.

Example 1: Zeros of a Polynomial

The zeros of a polynomial with real coefficients are computed.

```
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]);
            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.

```
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+1i
    radius = 6.11e-14
    status = False
root = 0.99999999999999856+2i
    radius = 1.95e-13
    status = False
root = 1.0000000000000013+3.0000000000000013i
    radius = 1.50e-13
    status = False
```

ZeroFunction Class

Summary

The ZeroFunction class uses Muller's method to find the zeros of a univariate function, f(x).

```
public class Imsl.Math.ZeroFunction
```

Properties

AbsoluteError

```
public double AbsoluteError {get; set; }
```

Description

The first stopping criterion.

A zero $x[i]$ is accepted if $|f(x[i])|$ is less than this tolerance. Its default value is about 1.0e-8.

MaximumIterations

```
public int MaximumIterations {get; set; }
```

Description

The maximum number of iterations allowed per root.

The default value is 100.

RelativeError

```
public double RelativeError {get; set; }
```

Description

The second stopping criterion is the relative error.

A zero $x[i]$ is accepted if the relative change of two successive approximations to $x[i]$ is less than this tolerance. Its default value is about 1.0e-8.

Spread

```
public double Spread {get; set; }
```

Description

The spread.

The default value is 1.0.

See Also: Imsl.Math.ZeroFunction.SpreadTolerance (p. 111)

SpreadTolerance

```
public double SpreadTolerance {get; set; }
```

Description

The spread criteria for multiple zeros.

If the zero $x[i]$ has been computed and $|x[i] - x[j]| <$ `SpreadTolerance`, where $x[j]$ is a previously computed zero, then the computation is restarted with a guess equal to $x[i]+$ `Spread`. The default value for `SpreadTolerance` is 1.0e-5.

Constructor

ZeroFunction

`public ZeroFunction()`

Description

Creates an instance of the solver.

Methods

AllConverged

`public bool AllConverged()`

Description

Returns `true` if the iterations for all of the roots have converged.

Returns

A `boolean` value specifying whether the roots have converged.

ComputeZeros

`public double[] ComputeZeros(Imsl.Math.ZeroFunction.IFunction f, double[] guess)`

Description

Returns the zeros of a univariate function.

Parameters

`f` – The `ZeroFunction.IFunction` to be integrated.

`guess` – A `double` array containing an initial guess of the zeros. A zero will be found for each point in guess.

Returns

A `double` array containing the zero of the univariate function.

GetIterations

`public int GetIterations(int nRoot)`

Description

Returns the number of iterations used to compute a root.

Parameter

`nRoot` – An `int` specifying the index of the root.

Returns

An `int` specifying the number of iterations necessary to compute a root.

Description

`ZeroFunction` computes n real zeros of a real function f. Given a user-supplied function $f(x)$ and an n-vector of initial guesses $x_1, x_2, \ldots, x_n$, the routine uses Muller's method to locate n real zeros of f, that is, n real values of x for which $f(x) = 0$. The routine has two convergence criteria. The first requires the absolute value of the function be less than the `AbsoluteError`. The second requires that the relative change of any two successive approximations to an x_i be less than `RelativeError`. Here, x_i^m is the m-th approximation to x_i. Let `AbsoluteError` be ε_1, and `RelativeError` be ε_2. The criteria may be stated mathematically as follows:

Criterion 1:

$$|f\ (x_i^m)| < \ \varepsilon_1$$

Criterion 2:

$$\left|\frac{x_i^{m+1} - x_i^m}{x_i^m}\right| < \ \varepsilon_2$$

"Convergence" is the satisfaction of either criterion.

Example: Zeros of a Univariate Function

In this example 3 zeros of the sin function are found.

```
using System;
using Imsl.Math;

public class ZeroFunctionEx1 : ZeroFunction.IFunction
{
    public double F(double x)
    {
        return Math.Sin(x);
    }


    public static void  Main(String[] args)
    {
        ZeroFunction.IFunction fcn = new ZeroFunctionEx1();

        ZeroFunction zf = new ZeroFunction();
        double[] guess = new double[]{5, 18, - 6};
        double[] zeros = zf.ComputeZeros(fcn, guess);
        for (int k = 0; k < zeros.Length; k++)
        {
            Console.Out.WriteLine
                (zeros[k] + " = " + (zeros[k] / Math.PI) + " pi");
        }
```

```
    }
}
```

Output

```
6.28318530717956 = 1.99999999999999 pi
18.8495559215629 = 6.0000000000077 pi
-6.28318530717964 = -2.00000000000002 pi
```

ZeroFunction.IFunction Interface

Summary

Interface for the user supplied function to ZeroFunction.

`public interface Imsl.Math.ZeroFunction.IFunction`

Method

F

`abstract public double F(double x)`

Description

The user supplied function to ZeroFunction.

Returns the value of the function at the given point.

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.

ZeroSystem Class

Summary

Solves a system of n nonlinear equations f(x) = 0 using a modified Powell hybrid algorithm.

`public class Imsl.Math.ZeroSystem`

Properties

MaximumIterations

```
public int MaximumIterations {get; set; }
```

Description

The maximum number of iterations allowed.

The default value is 200.

RelativeError

```
public double RelativeError {get; set; }
```

Description

The relative error tolerance.

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.

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 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.

See Also: Imsl.Math.ZeroSystem.IJacobian (p. 117)

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.

`Imsl.Math.TooManyIterationsException` id is thrown if the maximum number of iterations is exceeded

`Imsl.Math.ToleranceTooSmallException` id is thrown if the error tolerance is too small

`Imsl.Math.DidNotConvergeException` id is thrown if the algorithm does not converge

Description

`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`.

Example: Solve a System of Nonlinear Equations

A system of nonlinear equations is solved.

```
using System;
using Imsl.Math;

public class ZeroSystemEx1 : ZeroSystem.IFunction
{
    public void  F(double[] x, double[] f)
    {
        f[0] = x[0] + System.Math.Exp(x[0] - 1.0) + (x[1] + x[2]) *
            (x[1] + x[2]) - 27.0;
        f[1] = System.Math.Exp(x[1] - 2.0) / x[0] + x[2] * x[2] - 10.0;
        f[2] = x[2] + System.Math.Sin(x[1] - 2.0) + x[1] * x[1] - 7.0;
    }


    public static void  Main(String[] args)
    {
        ZeroSystem zf = new ZeroSystem(3);
        zf.SetGuess(new double[]{4, 4, 4});
        new PrintMatrix("zeros").Print(zf.Solve(new ZeroSystemEx1()));
```

```
    }
}
```

Output

```
        zeros
          0
0  0.99999999995498
1  2.00000000000656
2  2.99999999999468
```

ZeroSystem.IFunction Interface

Summary

Public interface for user supplied function to `ZeroSystem` object.

`public interface Imsl.Math.ZeroSystem.IFunction`

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

Summary

Public interface for user supplied function to `ZeroSystem` object.

```
public interface Imsl.Math.ZeroSystem.IJacobian :
Imsl.Math.ZeroSystem.IFunction
```

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

Usage Notes

Unconstrained Minimization

The unconstrained minimization problem can be stated as follows:

$$\min_{x \in R^n} f(x)$$

where $f : \mathbf{R}^n \to \mathbf{R}$ 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.Gradient`.

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:

$$\begin{array}{l} \min\limits_{x \in R^n} f(x) \\ \text{subject to } A_1 x = b_1 \end{array}$$

where $f : \mathbf{R}^n \to \mathbf{R}$, 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 x n matrix

$$\nabla^2 f(x) = \left[\frac{\partial^2}{\partial x_i \partial x_j} f(x) \right]$$

Nonlinearly Constrained Minimization

The nonlinearly constrained minimization problem can be stated as follows:

$$\begin{array}{l} \min\limits_{x \in R^n} f(x) \\ \text{subject to } \; g_i(x) = 0 \;\; \text{for } \; i = 1,\ 2,\ \ldots,\ m_1 \\ g_i(x) \geq 0 \;\; \text{for } \; i = m_1 + 1,\ \ldots,\ m \end{array}$$

where $f : \mathbf{R}^n \to \mathbf{R}$ and $g_i : \mathbf{R}^n \to \mathbf{R}$, 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.

MinUncon Class

Summary

Finds the minimum point for a smooth univariate function using function and optionally first derivative evaluations.

```
public class Imsl.Math.MinUncon
```

Properties

Accuracy

```
public double Accuracy {get; set; }
```

Description

The required absolute accuracy in the final value returned by the `ComputeMin` method.

By default, the required accuracy is set to 1.0e-8.

Bound

```
public double Bound {get; set; }
```

Description

The amount by which X may be changed from its initial value, `Guess`.

By default, `Bound` is set to 100.

DerivTolerance

`public double DerivTolerance {get; set; }`

Description

The derivative tolerance used by member method `ComputeMin` to decide if the current point is a local minimum.

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

`public double Guess {get; set; }`

Description

The initial guess of the minimum point of the input function.

By default, an initial guess of 0.0 is used.

Step

`public double Step {get; set; }`

Description

The stepsize to use when changing x.

By default, `Step` is set to 0.1.

Constructor

MinUncon

`public MinUncon()`

Description

Unconstrained minimum constructor for a smooth function of a single variable of type `double`.

Method

ComputeMin

`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.

Description

`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 $x_0 = Guess$ where `Guess` is specified by the `Guess` property and $b = Bound$, then x is restricted to the interval $[x_0 - b, x_0 + b]$. Usually, the algorithm begins the search by moving from x_0 to $x = x_0 + s$, 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 $x_0 \pm b$. 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, x_1, x_2, and x_3, with $x_1 < x_2 < x_3$ and $f(x_2) \le f(x_1)$ and $f(x_2) \le f(x_3)$. 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 x_2 is near the center of the range between x_1 and x_3 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 x_2, and subject to being in the longer interval between x_1 and x_2 or x_2 and x_3 when x_2 is particularly close to x_1 or x_3. 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 ε 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 δ of x, where $\delta = xacc$, 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:

$$|x_c - x_n| \leq \varepsilon_c$$

Criterion 2:

$$|g_c| \leq \varepsilon_g$$

where $\varepsilon_c = \max\{1.0, |x_c|\}\,\varepsilon$, ε is a relative error tolerance and ε_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.

Example 1: Minimum of a smooth function

The minimum of $e^x - 5x$ is found using function evaluations only.

```
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 1.60941759992003
```

Example 2: Minimum of a smooth function

The minimum of $e^x - 5x$ is found using function evaluations and first derivative evaluations.

```
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;
```

```
    }


    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

Summary

Interface for the user supplied function for the smooth function of a single variable to be minimized.

`public interface Imsl.Math.MinUncon.IFunction`

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

Summary

Interface for the smooth function of a single variable to be minimized and its derivative.

`public interface Imsl.Math.MinUncon.IDerivative :  Imsl.Math.MinUncon.IFunction`

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

Summary

Minimizes a multivariate function using a quasi-Newton method.

`public class Imsl.Math.MinUnconMultiVar`

Properties

Digits

`public double Digits {get; set; }`

Description

The number of good digits in the function.

By default, `Digits` is set to 15.75.

ErrorStatus

```
public int ErrorStatus {get; }
```

Description

The non-fatal error status.

FalseConvergenceTolerance

```
public double FalseConvergenceTolerance {get; set; }
```

Description

The false convergence tolerance.

By default, 2.22044604925031308e-14 is used as the false convergence tolerance.

Fscale

```
public double Fscale {get; set; }
```

Description

The function scaling value for scaling the gradient.

By default, the value of this scalar is set to 1.0.

GradientTolerance

```
public double GradientTolerance {get; set; }
```

Description

The gradient tolerance used to compute the gradient.

By default, the cube root of machine precision squared is used to compute the gradient.

Ihess

```
public int Ihess {get; set; }
```

Description

The Hessian initialization parameter.

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

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.

MaximumStepsize

```
public double MaximumStepsize {get; set; }
```

Description

The maximum allowable stepsize to use.

By default, maximum stepsize is set to a value based on a scaled `Guess`.

MaxIterations

```
public int MaxIterations {get; set; }
```

Description

The maximum number of iterations allowed.

By default, the maximum number of iterations is set to 100.

RelativeTolerance

```
public double RelativeTolerance {get; set; }
```

Description

The relative function tolerance.

By default, 3.66685e-11 is used as the relative function tolerance.

StepTolerance

```
public double StepTolerance {get; set; }
```

Description

The scaled step tolerance to use when changing x.

The i-th component of the scaled step between two points x and y is computed as

abs(x(i)-y(i))/max(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.

Constructor

MinUnconMultiVar

```
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

`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.

`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.

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.

`Imsl.Math.FalseConvergenceException` id is thrown if the iterates appear to be converging to a noncritical point

`Imsl.Math.MaxIterationsException` id is thrown if the maximum number of iterations is exceeded

`Imsl.Math.UnboundedBelowException` id 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

`public void SetGuess(double[] guess)`

Description

Sets the initial guess of the minimum point of the input function.

By default, the elements of this array are set to 0.0.

Parameter

`guess` – A `double` array specifying the initial guess of the minimum point of the input function.

SetXscale

`public void SetXscale(double[] xscale)`

Description

Sets the diagonal scaling matrix for the variables.

By default, the elements of this array are set to 1.0.

Parameter

`xscale` – A `double` array specifying the diagonal scaling matrix for the variables.

`System.ArgumentException` id is thrown if any of the elements of `Xscale` is less than or equal to or equal to 0

Description

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:

$$\min_{x \in R^n} f(x)$$

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

$$f(x_n) \leq f(x_c) + \alpha g^T d, \quad \alpha \in (0,\, 0.5)$$

Finally, the optimality condition $||\mathrm{g(x)}|| \leq \varepsilon$ where ε is a gradient tolerance.

When optimality is not achieved, B is updated according to the BFGS formula

$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

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`).

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.

```
using System;
using Imsl.Math;

public class MinUnconMultiVarEx1 : MinUnconMultiVar.IFunction
{
    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 static void  Main(String[] args)
    {
        MinUnconMultiVar solver = new MinUnconMultiVar(2);
        solver.SetGuess(new double[]{- 1.2, 1.0});
        double[] x = solver.ComputeMin(new MinUnconMultiVarEx1());
        Console.Out.WriteLine
            ("Minimum point is (" + x[0] + ", " + x[1] + ")");
    }
}
```

Output

```
Minimum point is (0.99999996726513, 0.99999993304521)
```

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.

```
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]);
    }


    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 (0.999999966882301, 0.999999932254245)
```

MinUnconMultiVar.IFunction Interface

Summary

Interface for the user supplied multivariate function to be minimized.

`public interface Imsl.Math.MinUnconMultiVar.IFunction`

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

Summary

Interface for the user supplied multivariate function to be minimized and its gradient.

```
public interface Imsl.Math.MinUnconMultiVar.IGradient :
Imsl.Math.MinUnconMultiVar.IFunction
```

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

Summary

Solves a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm.

```
public class Imsl.Math.NonlinLeastSquares
```

Properties

AbsoluteTolerance

```
public double AbsoluteTolerance {get; set; }
```

Description

The absolute function tolerance.

By default, 1.0e-32 is used as the absolute function tolerance.

Digits

```
virtual public int Digits {get; set; }
```

Description

The number of good digits in the function.

By default, the number of good digits is set to 7.

ErrorStatus

```
public int ErrorStatus {get; }
```

Description

Get information about the performance of NonlinLeastSquares.

Value	Meaning
0	All convergence tests were met.
1	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.
2	Scaled actual and predicted reductions in the function are less than or equal to the relative function convergence tolerance `RelativeTolerance`.
3	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.
4	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.

See Also: Imsl.Math.NonlinLeastSquares.RelativeTolerance (p. 136), Imsl.Math.NonlinLeastSquares.StepTolerance (p. 136)

FalseConvergenceTolerance

```
public double FalseConvergenceTolerance {get; set; }
```

Description

The false convergence tolerance.

By default, 100.0e-16 is used as the false convergence tolerance.

GradientTolerance

`public double GradientTolerance {get; set; }`

Description

The gradient tolerance used to compute the gradient.

By default, the cube root of machine precision squared is used to compute the gradient.

InitialTrustRegion

`public double InitialTrustRegion {get; set; }`

Description

The initial trust region radius.

By default, `InitialTrustRegion` is set based on the initial scaled Cauchy step.

MaximumIterations

`public int MaximumIterations {get; set; }`

Description

The maximum number of iterations allowed.

By default, the maximum number of iterations is set to 100.

MaximumStepsize

`public double MaximumStepsize {get; set; }`

Description

The maximum allowable stepsize to use.

By default, the maximum stepsize is set to a default value based on a scaled `Guess`.

RelativeTolerance

`public double RelativeTolerance {get; set; }`

Description

The relative function tolerance.

By default, 1.0e-20 is used as the relative function tolerance.

StepTolerance

`public double StepTolerance {get; set; }`

Description

The step tolerance used to step between two points.

By default, the cube root of machine precision is used as the step tolerance.

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.

By default, the identity is used.

Parameter

`fscale` – A `double` array specifying the diagonal scaling matrix for the functions.

`System.ArgumentException` id 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.

By default, an initial guess of 0.0 is used.

Parameter

`guess` – A `double` array specifying the initial guess of the minimum point of the input function.

SetXscale

`public void SetXscale(double[] xscale)`

Description

Set the diagonal scaling matrix for the variables.

By default, the identity is used.

Parameter

`xscale` – A `double` array specifying the diagonal scaling matrix for the variables.

`System.ArgumentException` id 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.

`Imsl.Math.TooManyIterationsException` id is thrown if the number of iterations exceeds `MaximumIterations`, `MaximumIterations` is set to 100 by default

Description

`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:

$$\min_{x \in R^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

where $m \geq n$, $F: \; R^n \rightarrow R^m$, and $f_i(x)$ is the i-th component function of *F(x)*. From a current point, the algorithm uses the trust region approach:

$$\min_{x_n \in R^n} \left\| F\left(x_c\right) + J\left(x_c\right)\left(x_n - x_c\right)\right\|_2$$

subject to

$$\left\| x_n - x_c \right\|_2 \leq \delta_c$$

to get a new point x_n, which is computed as

$$x_n = x_c - \left(J\left(x_c\right)^T J\left(x_c\right) + \mu_c I\right)^{-1} J\left(x_c\right)^T F\left(x_c\right)$$

where $\mu_c = 0$ if $\delta_c \geq \left\| \left(J\left(x_c\right)^T J\left(x_c\right)\right)^{-1} \; J\left(x_c\right)^T F\left(x_c\right)\right\|_2$ 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. 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`.

Example 1: Nonlinear least-squares problem

A nonlinear least-squares problem is solved using a finite-difference Jacobian.

```
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.

```
using System;
using Imsl.Math;

public 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

Summary

Interface for the user supplied nonlinear least-squares function.

`public interface Imsl.Math.NonlinLeastSquares.IFunction`

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

Summary

Interface for the user supplied nonlinear least squares function and its Jacobian.

```
public interface Imsl.Math.NonlinLeastSquares.IJacobian :
Imsl.Math.NonlinLeastSquares.IFunction
```

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

Summary

Solves a linear programming problem using an active set strategy.

`public class Imsl.Math.DenseLP : ICloneable`

Properties

IterationCount

`public int IterationCount {get; }`

Description

Returns the number of iterations used.

ObjectiveValue

`public double ObjectiveValue {get; }`

Description

Returns the optimal value of the objective function.

RefinementType

`public int RefinementType {get; set; }`

Description

The type of refinement used, if any.

The possible settings are:

Value	Action
0	No refinement. Always compute dual. Default.
1	Iterative refinement.
2	Use extended refinement. Iterate until no more progress.

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.

`System.ArgumentException` id 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.

`System.ArgumentException` id is thrown if the dimensions of `a`, `b.length`, and `c.length` are not consistent

Methods

Clone

`Final public Object Clone()`

Description

Creates and returns a copy of this object.

Returns

A copy of this object.

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.

Let $r_i = a_{i1}x_1 + \cdots + a_{in}x_n$

constraintType	Constraint
0	$\mathrm{r}_i = \mathrm{b}_i$
1	$\mathrm{r}_i \le \mathrm{b}_{u_i}$
2	$\mathrm{r}_i \ge \mathrm{b}_i$
3	$\mathrm{b}_i \le \mathrm{r}_i \le \mathrm{b}_{u_i}$
4	Ignore this constraint

Parameter

`constraintType` – A `int` array containing the types of general constraints.

SetLowerBound

`public void SetLowerBound(double[] lowerBound)`

Description

Sets the lower bound, x_l on the variables.

If there is no lower bound on a variable, then 10e30 should be set as the lower bound. Default = 0.

Parameter

`lowerBound` – A `double` array containing the lower bounds on the variables.

SetUpperBound

`public void SetUpperBound(double[] upperBound)`

Description

Sets the upper bound, x_u on the variables.

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.

Parameter

`upperBound` – A `double` array containing the upper bound on the variables.

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.

Solve must be invoked prior to any of the "get" methods.

`Imsl.Math.BoundsInconsistentException` id is thrown if the bounds are inconsistent

`Imsl.Math.NoAcceptablePivotException` id is thrown if an acceptable pivot could not be found.

`Imsl.Math.ProblemUnboundedException` id is thrown if there is no finite solution to the problem

Description

Class `DenseLP` uses an active set strategy to solve linear programming problems, i.e., problems of the form

$$\min_{x \in R^n} c^T x$$

subject to

$$b_l \le Ax \le b_u$$

$$x_l \le x \le x_u$$

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.

Refer to the following paper for further information: Krogh, Fred, T. (2005), An Algorithm for Linear Programming, http://mathalacarte.com/fkrogh/pub/lp.pdf , Tujunga, CA.

Example 1: Dense Linear Programming

The linear programming problem in the standard form

$$\min\ f(x) = -x_1 - 3x_2$$

subject to:

$x_1 + x_2 + x_3 = 1.5$
$x_1 + x_2 - x_4 = 0.5$
$x_1 + x_5 = 1.0$

$x_2 + x_6 = 1.0$
$x_i \geq 0, \quad \text{for } i = 1, \ldots, 6$

is solved.

```
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

```
Solution
    0
0  0.5
1  1
2  0
3  1
4  0.5
5  0
```

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 LinearProgrammingEx2
{
    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};

        LinearProgramming zf = new LinearProgramming(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
```

MPSReader Class

Summary

Reads a linear programming problem from an MPS file.

`public class Imsl.Math.MPSReader`

Fields

BINARY_VARIABLE
```
public int BINARY_VARIABLE
```

Description

Variable must be either 0 or 1.

CONTINUOUS_VARIABLE
```
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.

This is the value of the NAME field.

NameBounds
```
virtual public string NameBounds {get; set; }
```

Description

The name of the BOUNDS set.

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.

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.

An MPS file can contain multiple sets of RANGES, but only one is retained by this reader. If not set, then the first setin the file is used.

NameRHS

`virtual public string NameRHS {get; set; }`

Description

The name of the RHS set used.

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.

An binary constraint is the requirement that a variable be either 0 or 1. Binary constraints are also integer contraints.

NumberOfColumns

`virtual public int NumberOfColumns {get; }`

Description

The number of columns in the constraint matrix.

NumberOfIntegerConstraints

`virtual public int NumberOfIntegerConstraints {get; }`

Description

The number of integer constraints.

An integer constraint is the requirement that a variable be an integer.

NumberOfNonZeros

`virtual public int NumberOfNonZeros {get; }`

Description

The number of nonzeros in the constraint matrix.

NumberOfRows

`virtual public int NumberOfRows {get; }`

Description

The number of rows in the constraint matrix.

Objective

`virtual public Imsl.Math.MPSReader.Row Objective {get; }`

Description

The objective as a `Row`.

ObjectiveCoefficients

`virtual public double[] ObjectiveCoefficients {get; }`

Description

The coefficents of the objective row.

Constructor

MPSReader

`public MPSReader()`

Description

Initializes a new instance of the Imsl.Math.MPSReader (p. 148) class.

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.

Description

An MPS file defines a linear or quadratic programming problem. Linear programming problems read using this class are assumed to be of the form:

$$\min_{x \in R^n} c^T x$$

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`.

C	Objective
A	Constraint matrix
b_l	Lower Range
b_u	Upper Range
x_l	Lower Bound
x_u	Upper Bound

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:

Field Number	**Columns**	**Content**
1	2-3	Indicator
2	5-12	Name
3	15-22	Name
4	25-36	Value
5	40-47	Name
6	50-61	Value

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:

Row Type	**Meaning**
E	Equality constraint
L	Less than or equal constraint
G	Greater than or equal constraint
N	Objective of a free row

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.

Field	**Contents**
2	Column name
3	Row name
4	Value for the entry whose row and column are given by fields 2 and 3
5	Row name
6	Value for the entry whose row and column are given by fields 2 and 5

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.

Field	Contents
2	RHS name
3	Row name
4	Value for the entry whose row and column are given by fields 2 and 3
5	Row name
6	Value for the entry whose row and column are given by fields 2 and 5

Note: Fields 5 and 6 are optional.

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.

Field	Contents
2	Range set name
3	Row name
4	Value for the entry whose row and column are given by fields 2 and 3
5	Row name
6	Value for the entry whose row and column are given by fields 2 and 5

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:

Row Type	Lower Constraint	Upper Constraint
G	b_i	$b_i + \|r_i\|$
L	$b_i - \|r_i\|$	b_i
E	$b_i + min(0, r_i)$	$b_i + max(0, r_i)$

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 $2 \le x_i \le 6$ use a LO bound with value 2 to set $2 \le x_i$ and an UP bound with value 6 to add the condition $x_i \le 6$.

An MPS file can contain more than one bounds set, distinguished by the bound set name.

Field	**Contents**
1	Bounds type
2	Bounds set name
3	Column name
4	Value for the entry whose set and column are given by fields 2 and 3
5	Column name
6	Value for the entry whose set and column are given by fields 2 and 5

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.

Bound Type	**Definintion**	**Formula**
LO	Lower bound	$b_i \le x_i$
UP	Upper bound	$x_i \le b_i$
FX	Fixed Variable	$x_i = b_i$
FR	Free variable	$-\infty \le x_i \le \infty$
MI	Lower bound is minus infinity	$-\infty \le x_i$
PL	Upper bound is positive infinity	$x_i \le \infty$
BV	Binary variable (variable must be 0 or 1)	$x_i \in \{0, 1\}$
UI	Upper bound and integer	$x_i \le b_i$ and $x_i \in I$
LI	Lower bound and integer	$b_i \le x_i$ and $x_i \in I$
SC	Semicontinuous	0 or $b_i \le x_i$

The bound type names are not case sensitive.

If the bound type is UP or UI and $b_i \le x_i$ then the lower bound is set to $-\infty$.

ENDATA Section

The ENDATA section ends the MPS 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++)
        {
            System.Console.Out.WriteLine("   " + mps.GetLowerRange(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++)
        {
            System.Console.Out.WriteLine("   " + mps.GetLowerBound(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
```

LinearProgramming Class

Summary

Solves a linear programming problem using the revised simplex algorithm.

`public class Imsl.Math.LinearProgramming : ICloneable`

Properties

MaximumIterations

`public int MaximumIterations {get; set; }`

Description

Sets the maximum number of iterations. Default is set to 10000.

ObjectiveValue

`public double ObjectiveValue {get; }`

Description

Returns the optimal value of the objective function.

Constructor

LinearProgramming

`public LinearProgramming(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.

`System.ArgumentException` id is thrown if the dimensions of `a`, `b.length`, and `c.length` are not consistent

Methods

Clone

`Final public Object Clone()`

Description

Creates and returns a copy of this object.

Returns

A copy of this object.

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.

Let $r_i = a_{i1}x_1 + \cdots + a_{in}x_n$

constraintType	Constraint
0	$\mathrm{r}_i = \mathrm{b}_i$
1	$\mathrm{r}_i \leq \mathrm{bu}_i$
2	$\mathrm{r}_i \geq \mathrm{b}_i$
3	$\mathrm{b}_i \leq \mathrm{r}_i \leq \mathrm{bu}_i$

Parameter

`constraintType` – A `int` array containing the types of general constraints.

SetLowerBound

`public void SetLowerBound(double[] lowerBound)`

Description

Sets the lower bounds on the variables.

If there is no lower bound on a variable, then 10e30 should be set as the lower bound.

Parameter

`lowerBound` – A `double` array containing the lower bounds on the variables.

SetUpperBound

`public void SetUpperBound(double[] upperBound)`

Description

Sets the upper bound on the variables.

If there is no upper bound on a variable, then -10e30 should be set as the upper bound.

Parameter

`upperBound` – A `double` array containing the upper bound on the variables.

SetUpperLimit

`public void SetUpperLimit(double[] upperLimit)`

Description

Sets the upper limit of the constraints.

If no such constraint exists, then `bu` is not needed.

Parameter

`upperLimit` – A `double` array containing the upper limit of the constraints that have both the lower and the upper bounds.

Solve

`public void Solve()`

Description

Solves the problem using the revised simplex algorithm.

`Imsl.Math.BoundsInconsistentException` id is thrown if the bounds are inconsistent

`Imsl.Math.ProblemInfeasibleException` id is thrown if there is no feasible solution to the problem

`Imsl.Math.ProblemUnboundedException` id is thrown if there is no finite solution to the problem

`Imsl.Math.NumericDifficultyException` id is thrown if there is a numerical problem during the solution

Description

Class `LinearProgramming` uses a revised simplex method to solve linear programming problems, i.e., problems of the form

$$\min_{x \in R^n} c^T x$$

subject to

$$b_l \le A_x \le b_u$$

$$x_l \le x \le x_u$$

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.

For a complete description of the revised simplex method, see Murtagh (1981) or Murty (1983).

Example 1: Linear Programming

The linear programming problem in the standard form

$$\min\ f(x) = -x_1 - 3x_2$$

subject to:

$x_1 + x_2 + x_3 = 1.5$
$x_1 + x_2 - x_4 = 0.5$
$x_1 + x_5 = 1.0$
$x_2 + x_6 = 1.0$
$x_i \geq 0, \quad \text{for } i = 1, \ldots, 6$

is solved.

```
using System;
using Imsl.Math;

public class LinearProgrammingEx1
{
    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};

        LinearProgramming zf = new LinearProgramming(a, b, c);

        zf.Solve();
        new PrintMatrix("Solution").Print(zf.GetSolution());
    }
}
```

Output

```
Solution
    0
0  0.5
1  1
2  0
3  1
4  0.5
5  0
```

Example 2: Linear Programming

The linear programming problem

$$\min\; f(x) = -x_1 - 3x_2$$

subject to:

$0.5 \le x_1 + x_2 \le 1.5$
$0 \le x_1 \le 1.0$
$0 \le x_2 \le 1.0$

```
using System;
using Imsl.Math;

public class LinearProgrammingEx2
{
    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};

        LinearProgramming zf = new LinearProgramming(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
```

QuadraticProgramming Class

Summary

Solves the convex quadratic programming problem subject to equality or inequality constraints.

```
public class Imsl.Math.QuadraticProgramming
```

Property

NoMoreProgress

```
public bool NoMoreProgress {get; }
```

Description

Contains status of `true` or `false` if computer rounding error is inhibiting improvement in the objective function.

Usually the solution is close to optimum.

Constructor

QuadraticProgramming

```
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.

`Imsl.Math.InconsistentSystemException` id 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.

Description

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

$$\min_{x \in R^n} g^T x + \frac{1}{2} x^T H x$$

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.

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

```
  x
   0
0  1
1  1
```

```
2  1
3  1
4  1

          dual
            0
0   0
1  -1.18329135783152E-32
2   0
3   0
4   0
```

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

$$x_0 + 2x_1 - x_2 = 4$$

$$x_0 - x_1 + x_2 = -2$$

```
using System;
using Imsl.Math;

public class QuadraticProgrammingEx2
{
    public static void  Main(String[] args)
    {
        double[,] h = {
            {2, 0, 0},
            {0, 2, 0},
            {0, 0, 2}
        };
        double[,] aeq = {
            {1, 2, - 1},
            {1, - 1, 1}
        };
        double[] beq = new double[]{4, - 2};
        double[] g = new double[]{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

```
           x
           0
0   0.285714285714286
1   1.42857142857143
2  -0.857142857142857

         dual
           0
0   1.14285714285714
1  -0.571428571428572
2   0
```

MinConGenLin Class

Summary

Minimizes a general objective function subject to linear equality/inequality constraints.

`public class Imsl.Math.MinConGenLin`

Properties

FinalActiveConstraintsNum

`public int FinalActiveConstraintsNum {get; }`

Description

Returns the final number of active constraints.

ObjectiveValue

`public double ObjectiveValue {get; }`

Description

Returns the value of the objective function.

Tolerance

`public double Tolerance {get; set; }`

Description

The nonnegative tolerance on the first order conditions at the calculated solution.

Constructor

MinConGenLin

```
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`.

`System.ArgumentException` id 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.

`Imsl.Math.ConstraintsInconsistentException` id is thrown if the constraints are inconsistent.

`Imsl.Math.VarBoundsInconsistentException` id is thrown if the bounds on the variables are inconsistent.

`Imsl.Math.ConstraintsNotSatisfiedException` id is thrown if a solution satisfying the constraints could not be found.

`Imsl.Math.EqualityConstraintsException` id is thrown if the variables are determined by the constraints.

Description

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

$$A_1 x = b_1$$

$$A_2 x \le b_2$$

$$x_l \le x \le x_u$$

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

$$\min f\left(x^k\right) + d^T \nabla f\left(x^k\right) + \frac{1}{2} d^T B^k d$$

subject to

$$a_j d = 0,\ j \in I_k$$

$$a_j d \le 0,\ j \in J_k$$

is solved to get (d^k, λ^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 $x_i \le (x_u)_i$ and $a_j = -e_i$ for the constraint $-x_i \le (x_l)_i$. Here, e_i is a vector with 1 as the i-th component, and zeros elsewhere. Variables λ^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 α^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

$$\left(d^K\right)^T \nabla f\left(x^k + \alpha^k d^k\right) - \nabla f\left(x^k\right) > 0$$

holds. Let $x^k \leftarrow x^{k+1}$, and start another iteration.

The iteration repeats until the stopping criterion

$$\left\|\nabla f(x^k) - A^k \lambda^K\right\|_2 \leq \tau$$

is satisfied. Here τ is the supplied tolerance. For more details, see Powell (1988, 1989).

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)
    {
        return x[0] * x[0] + x[1] * x[1] + x[2] * x[2] + x[3] * x[3] +
            x[4] * x[4] - 2.0 * x[1] * x[2] - 2.0 * x[3] *
            x[4] - 2.0 * x[0];
    }

    public static void  Main(String[] args)
    {
        int neq = 2;
        int ncon = 2;
        int nvar = 5;
        double[] a = new double[]{1.0, 1.0, 1.0, 1.0, 1.0,
                                  0.0, 0.0, 1.0, - 2.0, - 2.0};
        double[] b = new double[]{5.0, - 3.0};
        double[] xlb = new double[]{0.0, 0.0, 0.0, 0.0, 0.0};
        double[] xub = new double[]{10.0, 10.0, 10.0, 10.0, 10.0};

        MinConGenLin.IFunction fcn = new MinConGenLinEx1();
        MinConGenLin zf = new MinConGenLin(fcn, nvar, ncon, neq, a, b,
            xlb, xub);
        zf.Solve();
        new PrintMatrix("Solution").Print(zf.GetSolution());
    }
}
```

Output

```
Solution
   0
0  1
1  1
2  1
3  1
4  1
```

Example 2: Linear Constrained Optimization

The problem

$$\min\ f(x) = -x_0 x_1 x_2$$

subject to

$$-x_0 - 2x_1 - 2x_2 \le 0$$

$$x_0 + 2x_1 + 2x_2 \le 72$$

$$0 \le x_0 \le 20$$

$$0 \le x_1 \le 11$$

$$0 \le x_2 \le 42$$

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

Summary

Public interface for the user-supplied function to evaluate the function to be minimized.

`public interface Imsl.Math.MinConGenLin.IFunction`

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

Summary

Public interface for the user-supplied function to compute the gradient.

```
public interface Imsl.Math.MinConGenLin.IGradient :
Imsl.Math.MinConGenLin.IFunction
```

Method

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

Summary

Solves a nonlinear least-squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm.

```
public class Imsl.Math.BoundedLeastSquares
```

Properties

AbsoluteTolerance

```
public double AbsoluteTolerance {get; set; }
```

Description

The absolute function tolerance.

Digits

```
public int Digits {get; set; }
```

Description

The number of good digits in the function.

GradientTolerance

```
public double GradientTolerance {get; set; }
```

Description

The scaled gradient tolerance.

MaximumFunctionEvals

```
public int MaximumFunctionEvals {get; set; }
```

Description

The maximum number of function evaluations.

MaximumIterations

```
public int MaximumIterations {get; set; }
```

Description

The maximum number of iterations.

MaximumJacobianEvals

```
public int MaximumJacobianEvals {get; set; }
```

Description

The maximum number of Jacobian evaluations.

MaximumStepsize

```
public double MaximumStepsize {get; set; }
```

Description

The maximum allowable step size.

RelativeTolerance

```
public double RelativeTolerance {get; set; }
```

Description

The relative function tolerance.

ScaledStepTolerance

```
public double ScaledStepTolerance {get; set; }
```

Description

The scaled step tolerance.

TrustRegion

`public double TrustRegion {get; set; }`

Description

The size of 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.

boundType	Action
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on first variable, all other variables will have the same bounds.

`lowerBound` – A `double` array containing the lower bounds on the variables.

`upperBound` – A `double` array containing the upper bounds on the variables.

`System.ArgumentException` id 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.

The i-th component of fscale is a positive scalar specifying the reciprocal magnitude of the i-th component function of the problem. Default: fscale[] = 1

Parameter

`fscale` – A `double` array containing the diagonal scaling for the functions.

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.

With this option, the values for `xscale` are set internally.

SetXscale

`public void SetXscale(double[] xscale)`

Description

The scaling vector for the variables.

Argument `xscale` is used mainly in scaling the gradient and the distance between two points. See `GradientTolernce` and `ScaledStepTolerance` for more details. Default: xscale[] = 1

Parameter

`xscale` – A `double` array containing the scaling vector for the variables.

---

solve

`public void solve()`

Description

Solves a nonlinear least-squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm.

`Imsl.Math.FalseConvergenceException` id is thrown if there is a problem with convergence.

Description

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:

$$min\frac{1}{2}F\left(x\right)^{T}F\left(x\right)=\frac{1}{2}\sum_{i=1}^{m}f_{i}\left(x\right)^{2}$$

subject to

$$l\leq x\leq u$$

here m $\geq$ n, $F:\mathrm{R}^{n}\rightarrow\mathrm{R}^{m}$, 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

$$d=-\left(J^{T}J+\mu I\right)^{-1}J^{T}F$$

where μ 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:

$$\|g(x_i)\| \le \varepsilon, l_i < x_i < u_i$$
$$g(x_i) < 0, x_i = u_i$$
$$g(x_i) > 0, x_i = l_i$$

where ε 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).

Example 1: Bounded Least Squares

The nonlinear least-squares problem

$$\min \frac{1}{2}\sum_{i=0}^{1} f_i(x)^2$$

$$-2 \le x_0 \le 0.5$$

$$-1 \le x_1 \le 2$$

where

$$f_0(x) = 10(x_1 - x_0^2) \text{ and } f_1(x) = (1 - x_0)$$

is solved.

```
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
0  0.5
1  0.250000000009201
```

Example 2: Bounded Least Squares

The nonlinear least-squares problem

$$\min \frac{1}{2} \sum_{i=0}^{1} f_i\left(x\right)^2$$

$$-2 \leq x_0 \leq 0.5$$

$$-1 \leq x_1 \leq 2$$

where

$$f_0(x) = 10(x_1 - x_0^2) \text{ and } f_1(x) = (1 - x_0)$$

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.

```
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;
    }


    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.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

Summary

Public interface for the user-supplied function to evaluate the function that defines the least-squares problem.

`public interface Imsl.Math.BoundedLeastSquares.IFunction`

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

Summary

Public interface for the user-supplied function to compute the Jacobian.

`public interface Imsl.Math.BoundedLeastSquares.IJacobian : Imsl.Math.BoundedLeastSquares.IFunction`

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.

MinConNLP Class

Summary

General nonlinear programming solver.

`public class Imsl.Math.MinConNLP`

Properties

BindingThreshold

`public double BindingThreshold {get; set; }`

Description

The binding threshold for constraints.

In the initial phase of minimization a constraint is considered binding if

$$\frac{g_i(x)}{max(1,\|\nabla g_i(x)\|)} \leq BindingThreshold \qquad i = M_e + 1, \ldots, M$$

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`.

BoundViolationBound

`public double BoundViolationBound {get; set; }`

Description

The amount by which bounds may be violated during numerical differentiation.

By default, `BoundViolationBound` is set to 1.0.

DifferentiationType

`public int DifferentiationType {get; set; }`

Description

The type of numerical differentiation to be used.

FunctionPrecision

`public double FunctionPrecision {get; set; }`

Description

The relative precision of the function evaluation routine.

By default, `FunctionPrecision` is set to 2.2e-16.

GradientPrecision

`public double GradientPrecision {get; set; }`

Description

The relative precision in gradients.

By default, `GradientPrecision` is set to 2.2e-16.

MaximumIterations

`public int MaximumIterations {get; set; }`

Description

The maximum number of iterations allowed.

By default, `MaximumIterations` is set to 200.

MultiplierError

`public double MultiplierError {get; set; }`

Description

The error allowed in the multipliers.

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 $e^{2\log\epsilon/3}$.

PenaltyBound

`public double PenaltyBound {get; set; }`

Description

The universal bound for describing how much the unscaled penalty-term may deviate from zero.

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.

ScalingBound

`public double ScalingBound {get; set; }`

Description

The scaling bound for the internal automatic scaling of the objective function.

By default, `ScalingBound` is set to 1.0e4.

ViolationBound

`public double ViolationBound {get; set; }`

Description

Defines allowable constraint violations of the final accepted result.

Constraints are satisfied if $|g_i(x)| \leq ViolationBound$, and $g_i(x) \geq -ViolationBound$ respectively. By default, `ViolationBound` is set to $min(\texttt{BindingThreshold}/10, max(epsdif, min(\texttt{BindingThreshold}/10, max((1.e-6)\texttt{BindingThreshold}, small_w)))$.

Constructor

MinConNLP

```
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

`public double[] GetConstraintResiduals()`

Description

Returns the constraint residuals.

Returns

A `double` array containing the constraint residuals.

GetLagrangeMultiplierEst

`public double[] GetLagrangeMultiplierEst()`

Description

Returns the Lagrange multiplier estimates of the constraints.

Returns

A `double` array containing the Lagrange multiplier estimates of the constraints.

SetGuess

`public void SetGuess(double[] guess)`

Description

Sets the initial guess of the minimum point of the input function.

By default, the elements of this array are set to x, (with the smallest value of $\|x\|_2$) that satisfies the bounds.

Parameter

`guess` – A `double` array specifying the initial guess of the minimum point of the input function.

SetXlowerBound

`public void SetXlowerBound(double[] lower)`

Description

Sets the lower bounds on the variables.

By default, the elements of this array are set to -1.79e308.

Parameter

`lower` – A `double` array specifying the lower bounds on the variables.

SetXscale

`public void SetXscale(double[] scale)`

Description

The internal scaling of the variables.

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.

Parameter

`scale` – A `double` array specifying the internal scaling of the variables.

`System.ArgumentException` id 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.

By default, the elements of this array are set to 1.79e308.

Parameter

`upper` – A `double` array specifying the upper bounds on the variables.

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 solution of the nonlinear programming problem.

`Imsl.Math.ConstraintEvaluationException` id is thrown if a constraint evaluation returns an error.

`Imsl.Math.ObjectiveEvaluationException` id is thrown if objective evaluation returns an error.

`Imsl.Math.WorkingSetSingularException` id is thrown if

`Imsl.Math.QPInfeasibleException` id is thrown if the working set is singular in dual extended QP.

`Imsl.Math.PenaltyFunctionPointInfeasibleException` id is thrown if the penalty function point infeasible.

`Imsl.Math.LimitingAccuracyException` id is thrown if limiting accuracy reached for a singular problem.

`Imsl.Math.TooManyIterationsException` id is thrown if maximum number of iterations exceeded.

`Imsl.Math.NoAcceptableStepsizeException` id is thrown if there is no acceptable stepsize.

`Imsl.Math.BadInitialGuessException` id is thrown if the penalty function point infeasible for original problem.

`Imsl.Math.IllConditionedException` id is thrown if the problem is singular or ill-conditioned.

`Imsl.Math.SingularException` id is thrown ifthe problem is singular.

`Imsl.Math.LinearlyDependentGradientsException` id is thrown if the working set gradients are linearly dependent.

`Imsl.Math.TerminationCriteriaNotSatisfiedException` id is thrown if termination criteria are not satisfied.

Description

`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 Armjijo-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 subproblems.* Math. Prog. 82, (1998), 413-448.

P. Spellucci: *A new technique for inconsistent problems in the SQP method.* Math. Meth. of Oper. Res. 47, (1998), 355-500. (published by Physica Verlag, Heidelberg, Germany).

The problem is stated as follows:

$$\min_{x \in R^n} f(x)$$

subject to

$$g_j(x) = 0, \text{for } j = 1, \ldots, m_e$$

$$g_j(x) \geq 0, \text{for j} = \mathrm{m_e} + 1, \ldots, \mathrm{m}$$

$$x_l \leq x \leq x_u$$

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 $l_i \le g_i(x) \le u_i$ is transformed into two constraints, $g_{2i}(x) \ge 0$ and $g_{2i+1}(x) \ge 0$, 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.

Example 1: Solving a general nonlinear programming problem

A general nonlinear programming problem is solved using a finite difference gradient.

```
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] - 2e0) +
                (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

```
          x
           0
0  0.822875655532512
1  0.911437827766256
```

Example 2: Solving a general nonlinear programming problem

A general nonlinear programming problem is solved using a user-supplied gradient.

```
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  0.822875655532512
1  0.911437827766256
```

MinConNLP.IFunction Interface

Summary

Public interface for the user supplied function to the `MinConNLP` object.

`public interface Imsl.Math.MinConNLP.IFunction`

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 `boolean` 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

Summary

Public interface for the user supplied function to compute the gradient for `MinConNLP` object.

`public interface Imsl.Math.MinConNLP.IGradient :  Imsl.Math.MinConNLP.IFunction`

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.

Chapter 9: Special Functions

Types

Sfun Class

Summary

Collection of special functions.

`public class Imsl.Math.Sfun`

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.

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.

Parameters

`a` – A `double` value.

`b` – A `double` value.

Returns

A `double` value specifying the Beta function.

BetaIncomplete

```
static public double BetaIncomplete(double x, double p, double q)
```

Description

Returns the incomplete Beta function ratio.

The incomplete beta function is defined to be

$$I_x(p,\ q) = \frac{\beta_x(p,\ q)}{\beta(p,\ q)} = \frac{1}{\beta(p,\ q)}\int_0^x t^{p-1}(1-t)^{q-1}dt \text{ for } 0 \le x \le 1,\ p > 0,\ q > 0$$

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).

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.

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)`

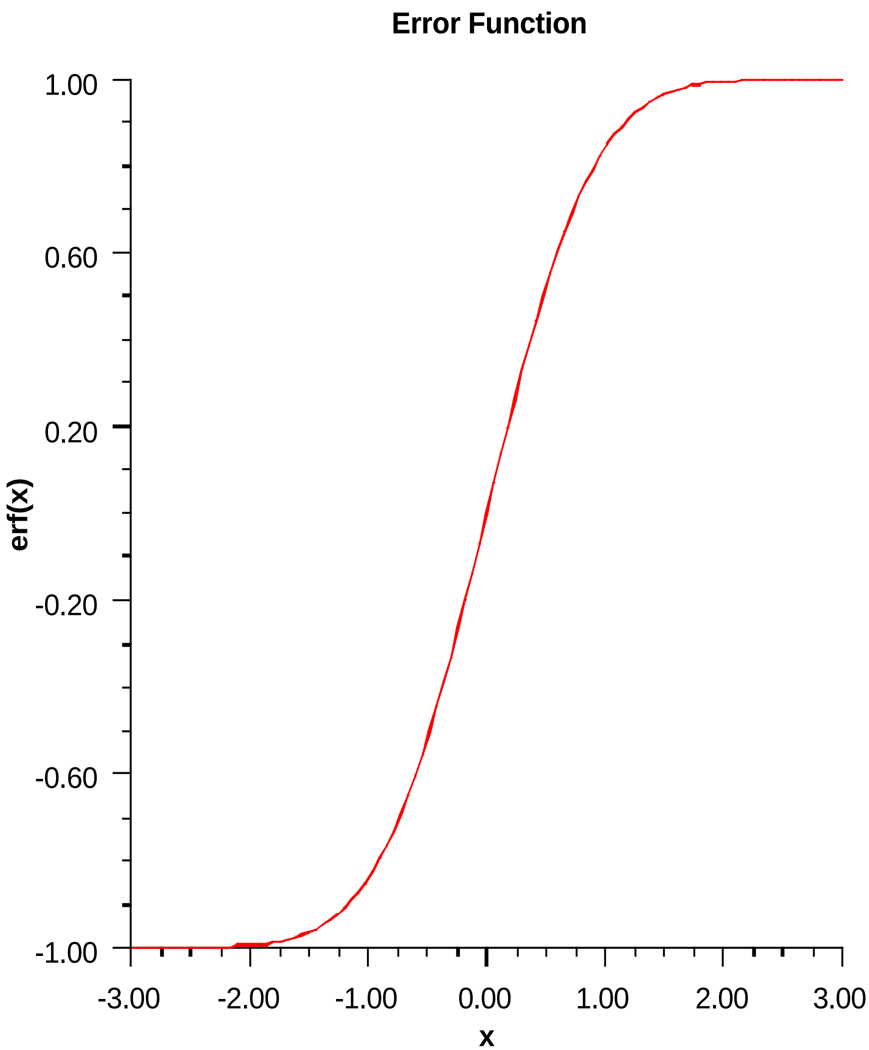
Description

Returns the error function of a `double`.

The error function method, `Erf`(x), is defined to be

$$\mathrm{erf}\,(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

All values of x are legal.



Parameter

`x` – A `double` value.

Returns

A `double` value specifying the error function of x.

Erfc

```
static public double Erfc(double x)
```

Description

Returns the complementary error function of a `double`.

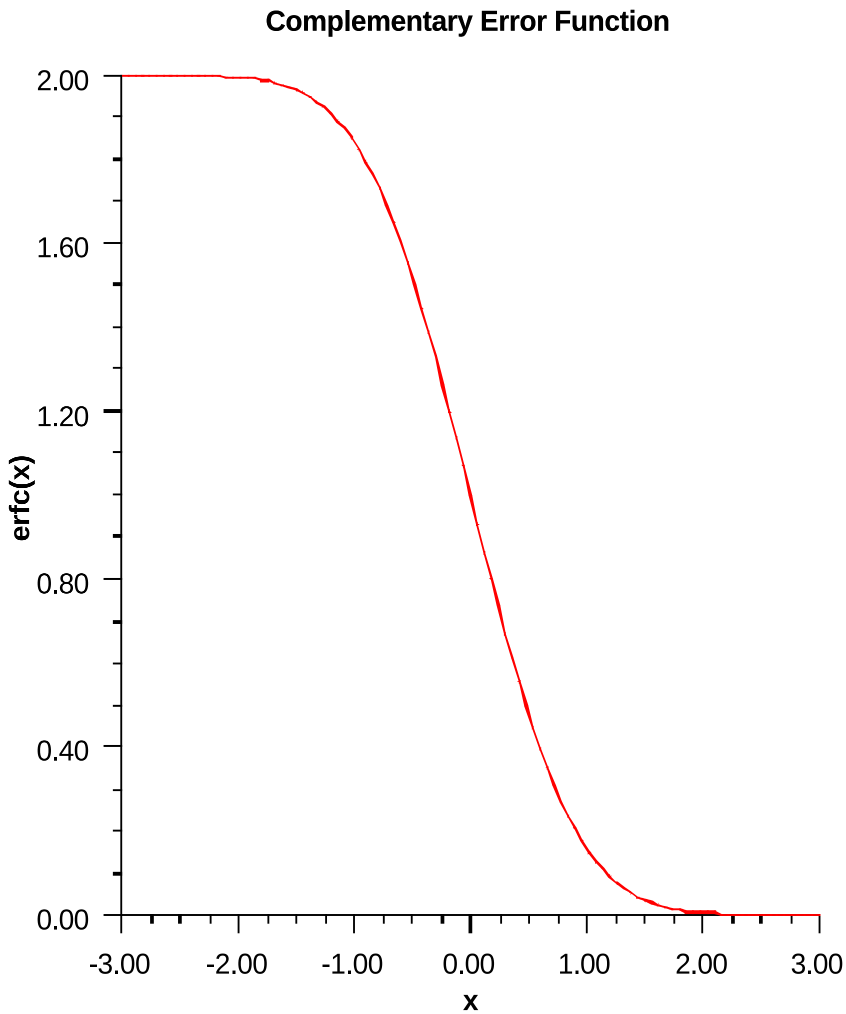
The complementary error function method, `Erfc` (x), is defined to be

$$\mathrm{erfc}\,(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$$

The argument x must not be so large that the result underflows. Approximately, x should be less than

$$\left[-ln\left(\sqrt{\pi}s\right)\right]^{1/2}$$

where $s = Double.Epsilon$ is the smallest representable positive floating-point number.



Parameter

x – A double value.

Returns

A double value specifying the complementary error function of x.

ErfcInverse

```
static public double ErfcInverse(double x)
```

Description

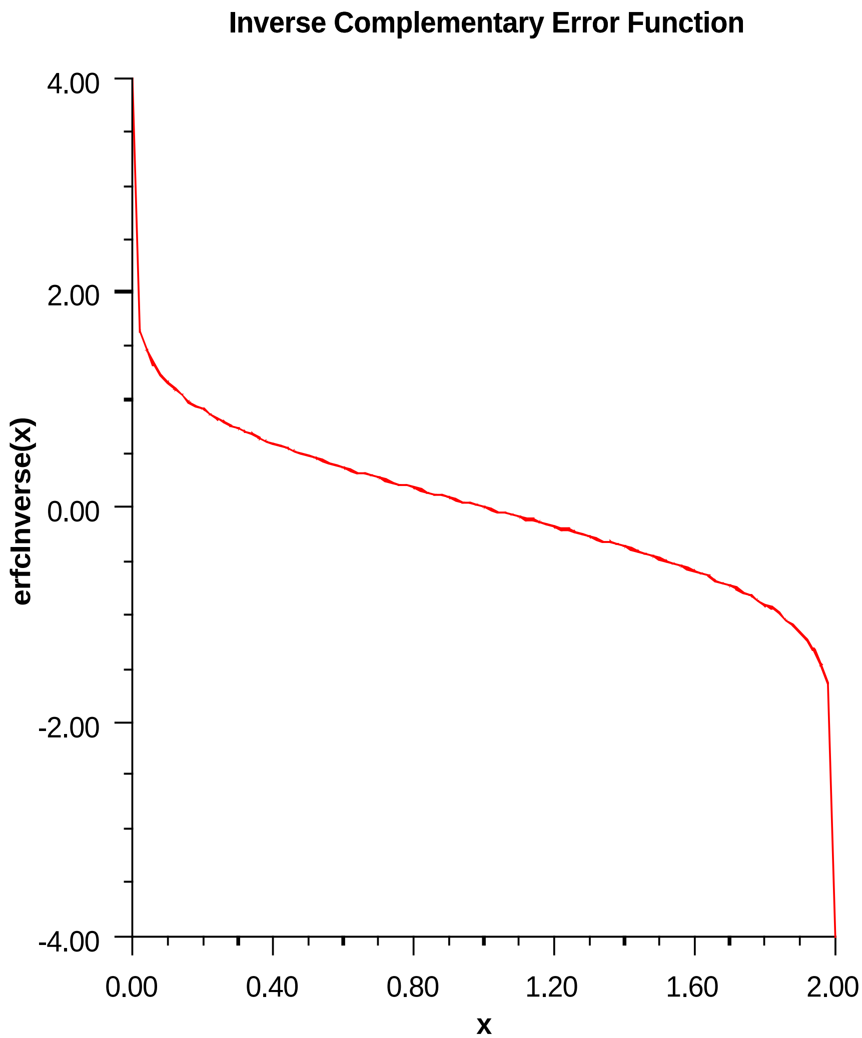
Returns the inverse of the complementary error function.

The `Erfcinverse(x)` method computes the inverse of the complementary error function erfc x, defined in `Erfc`.

`Erfcinverse(x)` is defined for $0 < x < 2$. If $x_{\max} < x < 2$, then the answer will be less accurate than half precision. Very approximately,

$$x_{max} \approx 2 - \sqrt{\varepsilon/(4\pi)}$$

where ε = machine precision (approximately 1.11e-16).



Parameter

x – A double value, $0 \leq x \leq 2$.

Returns

A double value specifying the inverse of the error function of x.

ErfInverse

```
static public double ErfInverse(double x)
```

Description

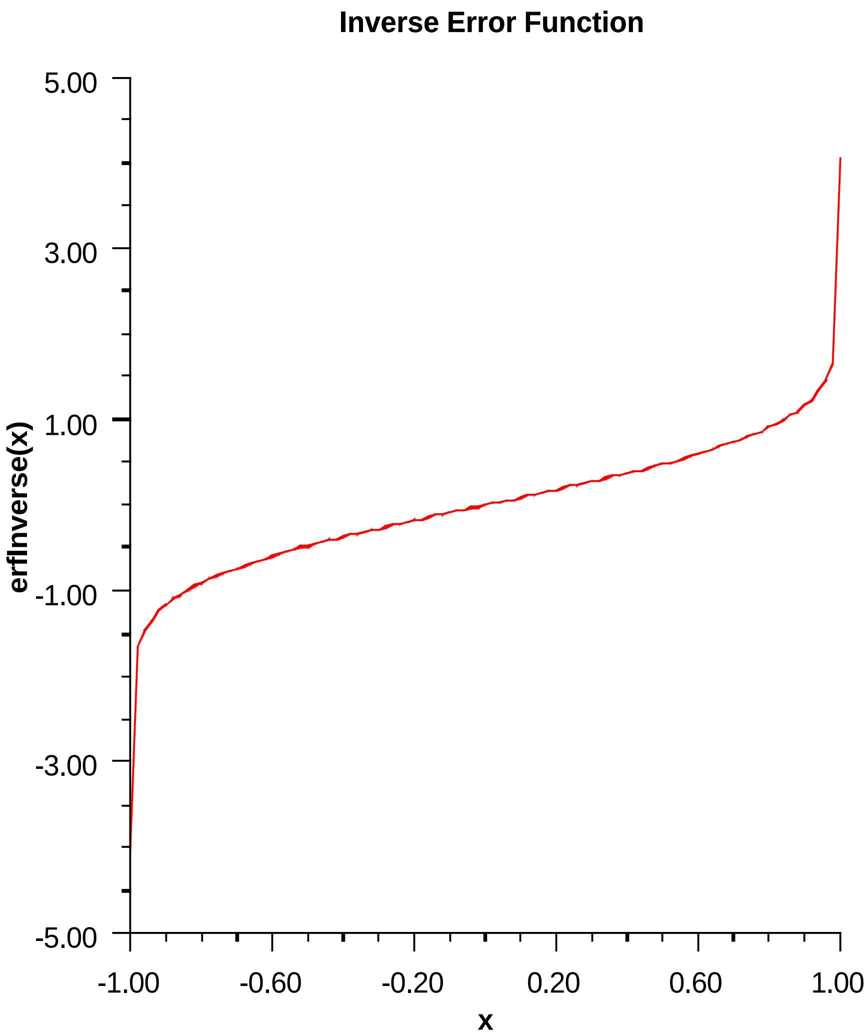
Returns the inverse of the error function.

`ErfInverse(X)` method computes the inverse of the error function erf x, defined in `Erf`.

The method `ErfInverse(X)` is defined for $x_{max} < |x| < 1$, then the answer will be less accurate than half precision. Very approximately,

$$x_{\max} \approx 1 - \sqrt{\varepsilon / (4\pi)}$$

where ε is the machine precision (approximately 1.11e-16).



Parameter

x – A double value.

Returns

A double value specifying the inverse of the error function of x.

Fact

`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

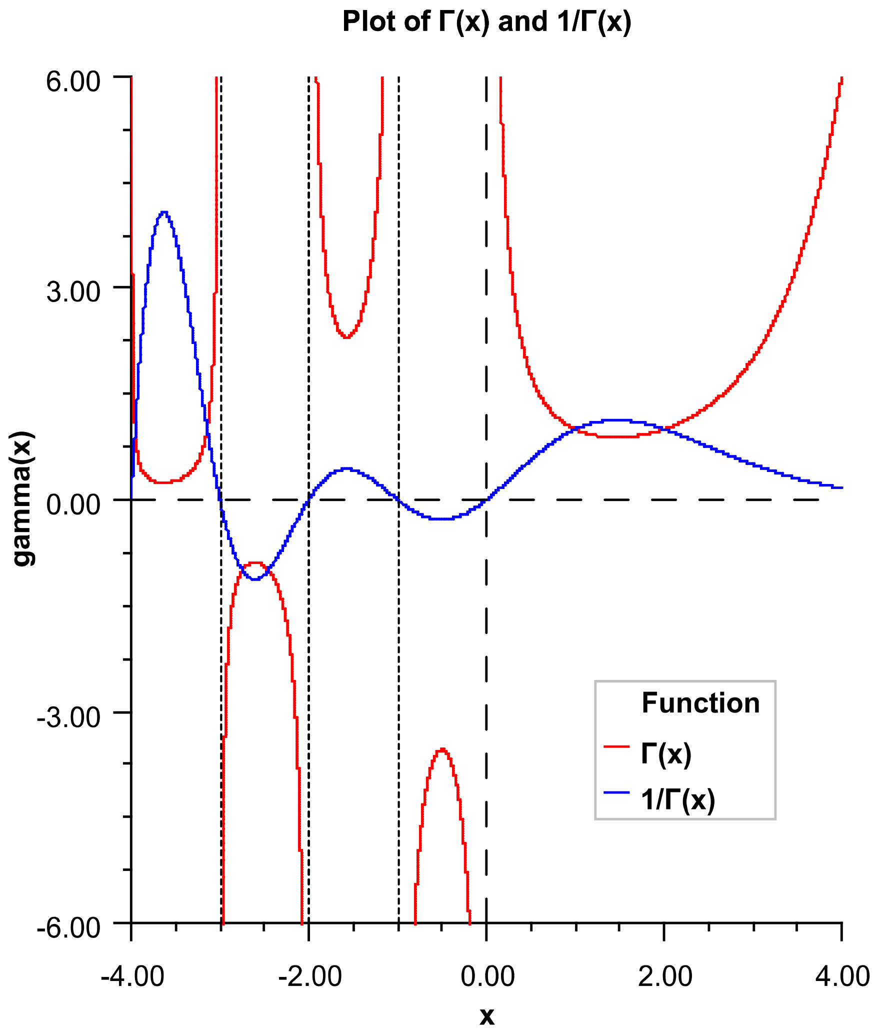
Returns the Gamma function of a `double`.

The Gamma function, $\Gamma(x)$, is defined to be

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \quad for\, x > 0$$

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.



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.

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 plus 1).

Specifically:

Log1p(± 0) returns ± 0.

Log1p(-1) returns $-\infty$.

Log1p(x) returns NaN, if $x < -1$.

Log1p($\pm\infty$) returns $\pm\infty$.

Parameter

`x` – A `double` value representing the argument.

Returns

A `double` value representing Log(1+x).

LogBeta

```
static public double LogBeta(double a, double b)
```

Description

Returns the logarithm of the Beta function.

Method `LogBeta` computes $\ln \beta(a, b) = \ln \beta(b, a)$. See `Beta` for the definition of $\beta(a, b)$.

`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.

Parameters

`a` – A `double` value.

`b` – A `double` value.

Returns

A `double` value specifying the natural logarithm of the Beta function.

LogGamma

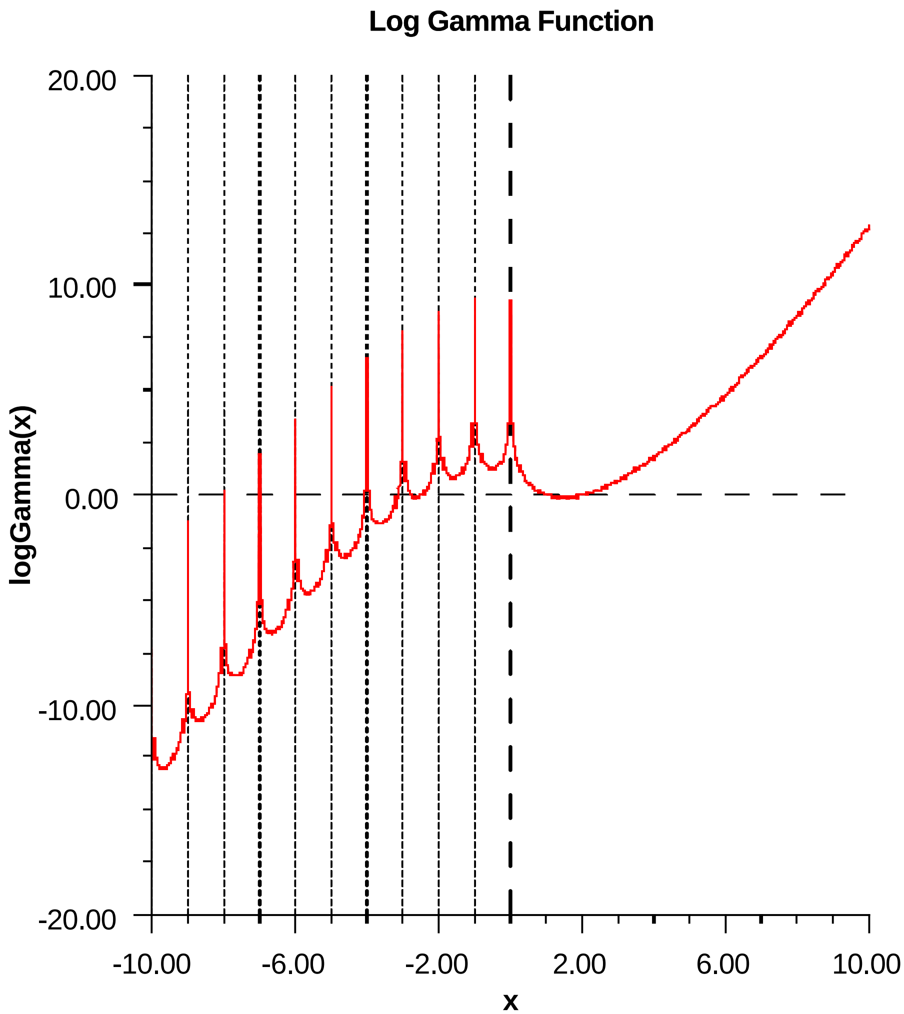
```
static public double LogGamma(double x)
```

Description

Returns the logarithm of the Gamma function of the absolute value of a `double`.

Method `LogGamma` computes $\ln|\Gamma(x)|$. See `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.



Parameter

`x` – A `double` value.

Returns

A `double` value specifying the natural logarithm of the Gamma function of $|x|$. If x is a negative integer, the result is NaN.

Poch

`static public double Poch(double a, double x)`

Description

Returns a generalization of Pochhammer's symbol.

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.

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).

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;

public 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 \n    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

Summary

Collection of Bessel functions.

`public class Imsl.Math.Bessel`

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.

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.

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.

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.

Bessel.I[i] contains the value of the Bessel function of order `i+xnu`. The Bessel function $I_v(x)$, is defined to be

$$I_\nu(x) = \frac{1}{\pi}\int_0^\pi e^{x\cos\theta}\cos(\nu\theta)d\,\theta - \frac{\sin(\nu\pi)}{\pi}\int_0^\infty e^{-x\cosh t - vt}dt$$

Here, argument `xnu` is represented by ν 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 $\nu =$ `xnu` must satisfy $0 \leq \nu \leq 1$.

This function is based on a code due to Cody (1983), which uses backward recursion.

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.

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.

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.

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.

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.

The Bessel function $J_v(x)$, is defined to be

$$J_\nu(x) = \frac{(x/2)^\nu}{\sqrt{\pi}\Gamma(\nu+1/2)}\int_0^\pi \cos\,(x\;\cos\,\theta)\sin^{2\nu}\theta\;d\,\theta$$

This code is based on the work of Gautschi (1964) and Skovgaard (1975). It uses backward recursion.

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`.

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.

This function uses $e^x K_{\nu+k-1}$ for $k = 1, \ldots, n$ and $\nu = 0$. For the definition of $K_v(x)$, see below.

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.

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.

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

$$K_\nu(x) = \frac{\pi}{2} e^{\nu\pi i/2} \left[i\, J_\nu(ix) - Y_\nu(ix) \right] \quad \text{for} -\pi < \text{arg x} \leq \frac{\pi}{2}$$

Currently, `xnu` (represented by ν 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`$[1] = K_{v+1}(x)$, ..., `K` $[n-1] = K_{v+n-1}(x)$.

This method is based on the work of Cody (1983).

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.

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.

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 $e^x K_{\nu+i-1}(x)$, 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. $|\nu|$ must be less than 1. Also, when $|n|$ is large compared with x, $|v + n|$ must not be so large that

$$e^x K_{\nu+n}(x) = e^x \frac{\Gamma(|\nu + n|)}{2(x/2)^{\nu+n}}$$

overflows. The code is based on work of Cody (1983).

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.

Y

```
static public double[] Y(double xnu, double x, int n)
```

Description

Evaluate a sequence of Bessel functions of the second kind with real nonnegative order and real positive argument.

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

$$Y_\nu(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin\theta - \nu\theta) d\,\theta$$

$$-\frac{1}{\pi}\int_0^\infty \left[e^{\nu t} + e^{-\nu t}\cos(\nu\pi)\right] e^{-x\sinh t}\,dt$$

The variable `xnu` (represented by ν in the above equation) must satisfy $0 \leq \nu < 1$. If this condition is not met, then `Y` is set to `NaN`. In addition, `x` must be in $[x_m, x_M]$ where $x_m = 6(16^{-32})$ 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.

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.

Example: The Bessel Functions

The Bessel functions I, J, and K are exercised for orders 0, 1, 2, and 3 at argument 10.e0.

```
using System;
using Imsl.Math;

public 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);

        Console.Out.WriteLine("Order     Bessel.I               " +
                              "Bessel.J              Bessel.K");
        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     1.77800623161676E-05
1     2670.98830370126     0.0434727461688615     1.86487734538256E-05
2     2281.51896772601     0.254630313685121     2.15098170069328E-05
3     1758.38071661085     0.0583793793051867     2.72527002565987E-05
```

Chapter 10: Miscellaneous

Types

Complex Structure

Summary

Set of mathematical functions for complex numbers. It provides the basic operations (addition, subtraction, multiplication, division) as well as a set of complex functions.

```
public structure Imsl.Math.Complex :  System.IComparable, System.IFormattable
```

Field

```
I
public Imsl.Math.Complex I
```

Description

The imaginary unit.

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.

`System.NullReferenceException` id 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.

19

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 = 0.195552704020373950+0.714335671546130541i
Root2 = -0.810937319404989250+0.208741251530792511i
```

Physical Structure

Summary

Return the value of various mathematical and physical constants.

`public structure Imsl.Math.Physical`

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.

12

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 convertions and computes the current units for the result.

```
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

Summary

The class is used to determine the limit of a sequence of approximations, by means of the

Epsilon algorithm of P. Wynn.

```
public class Imsl.Math.EpsilonAlgorithm
```

Property

ErrorEstimate

```
public double ErrorEstimate {get; }
```

Description

Returns 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.

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.

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.

Parameter

`maxTableSize` – A `int` which specifies the maximum size of Episilon Table to be computed.

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.

Description

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.

Chapter 11: Printing Functions

Types

PrintMatrix Class

Summary

Matrix printing utilities.

`public class Imsl.Math.PrintMatrix`

Constructors

PrintMatrix

`public PrintMatrix()`

Description

Creates an instance of the `PrintMatrix` class without a title and directs it to the default output stream.

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.

The matrix is printed without a title to `System.Console.Out`.

Parameter

`title` – A `String` which specifies the title to be printed above the matrix.

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

`void Print(string text)`

Description

Prints a string.

This function can be overridden to print to something other than a `PrintStream`.

Parameter

`text` – The `String` to be printed.

Print

```
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

```
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

```
void Println()
```

Description

Prints a newline.

This function can be overridden to print to something other than a `PrintStream`.

SetColumnSpacing

```
public Imsl.Math.PrintMatrix SetColumnSpacing(int columnSpacing)
```

Description

Sets the number of spaces between columns.

The default value is 2.

Parameter

`columnSpacing` – An `int` specifying the number of spaces between columns.

Returns

The `PrintMatrix` object.

SetEqualColumnWidths

```
public Imsl.Math.PrintMatrix SetEqualColumnWidths(bool equalColumnWidths)
```

Description

Force all of the columns to have the same width.

Parameter

`equalColumnWidths` – A `boolean` 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.

Values for `matrixType` are:

Value	Enumeration
0	MatrixType.Full
1	MatrixType.UpperTriangular
2	MatrixType.LowerTriangular
3	MatrixType.StrictUpperTriangular
4	MatrixType.StrictLowerTriangular

Parameter

`matrixType` – An `int` specifying the matrix type.

Returns

The `PrintMatrix` object.

SetPageWidth

```
public Imsl.Math.PrintMatrix SetPageWidth(int pageWidth)
```

Description

Sets the page width.

The default value is the largest possible integer.

Parameter

`pageWidth` – An `int` specifying the page width.

Returns

The `PrintMatrix` object.

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
   0  1  2  3
0  0  1  2  3
1  4  5  6  7
2  8  9  8  1
3  6  3  4  3

The 1 norm of the matrix is 20
```

PrintMatrixFormat Class

Summary

This class can be used to customize the actions of PrintMatrix.

`public class Imsl.Math.PrintMatrixFormat`

Properties

FirstColumnNumber

`public int FirstColumnNumber {get; set; }`

Description

Turns on column labeling with index numbers and sets the index for the label of the first column.

This is usually 0 or 1. The default is 0.

FirstRowNumber

`public int FirstRowNumber {get; set; }`

Description

Turns on row labeling with index numbers and sets the index for the label of the first row.

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 (p. 223) entries.

Constructor

PrintMatrixFormat

`public PrintMatrixFormat()`

Description

Constructs a `PrintMatrixFormat` object.

Methods

Format

```
virtual public string Format(Imsl.Math.PrintMatrixFormat.FormatType type,
  Object entry, int row, int col, Imsl.Math.PrintMatrixFormat.ParsePosition
  pos)
```

Description

Returns a formatted string.

Note, if `type` is not `FormatType.Entry`, *pos* will be set based on the following criteria.

entry	behavior
`double`	The index is the position of the decimal point.
`int`	The index is the position of the end of the formatted integer.

See Also: Imsl.Math.PrintMatrixFormat.FormatType (p. 239)

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. 241). 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. 241).

Returns

A `String` to be put into the printed table.

SetColumnLabels

`public void SetColumnLabels(string[] columnLabels)`

Description

Turns on column labeling using the given labels.

Parameter

`columnLabels` – An array of `String`s 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.

Description

By default, entries are formatted using the data type's `ToString` method.

See Also

Imsl.Math.PrintMatrix (p. 229)

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
```

PrintMatrixFormat.ParsePosition Class

Summary

Tracks the current position during parsing.

```
public class Imsl.Math.PrintMatrixFormat.ParsePosition
```

Property

Index

```
public int Index {get; set; }
```

Description

Current parse position.

Constructor

ParsePosition

```
public ParsePosition(int index)
```

Description

Creates a `ParsePosition`.

Parameter

`index` – The intial position.

PrintMatrix.MatrixType Enumeration

Summary

MatrixType indicates what part of the matrix is to be printed.

```
public enumeration Imsl.Math.PrintMatrix.MatrixType
```

Fields

`Full`

```
public Imsl.Math.PrintMatrix.MatrixType Full
```

Description

Indicates that the full matrix is to be printed.

`LowerTriangular`

```
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`

`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`

`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`

`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

Summary

FormatType specifies the argument to format.

`public enumeration Imsl.Math.PrintMatrixFormat.FormatType`

Fields

`BeginColumnLabel`

`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`

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

Summary

Type for column labels.

`public enumeration Imsl.Math.PrintMatrixFormat.ColumnLabelType`

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 `ints`.

`LabelString`
`public Imsl.Math.PrintMatrixFormat.ColumnLabelType LabelString`

Description

Specifies column labels will be an array of `Strings`.

PrintMatrixFormat.RowLabelType Enumeration

Summary

Type for row labels.

`public enumeration Imsl.Math.PrintMatrixFormat.RowLabelType`

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

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

Summary

Computes basic univariate statistics.

`public class Imsl.Stat.Summary`

Constructor

Summary

`public Summary()`

Description

Constructs a new summary statistics object.

Methods

GetConfidenceMean

`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

`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.

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

`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

`public double GetMinimum()`

Description

Returns the minimum.

Returns

A `double` representing the minimum.

GetMinimum

`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.

GetMode

`static public double GetMode(double[] x)`

Description

Returns the mode of the given data set.

Ties are broken at random.

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.

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.

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

`x` – 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.

Description

For the data in `x`, `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,

$$s_w^2 = \frac{\sum f_i w_i \left(x_i - \bar{x}_w\right)^2}{n-1}$$

Skewness,

$$\frac{\sum f_i w_i \left(x_i - \bar{x}_w\right)^3 / n}{[\sum f_i w_i \left(x_i - \bar{x}_w\right)^2 / n]^{3/2}}$$

Excess or Kurtosis,

$$\frac{\sum f_i w_i \left(x_i - \bar{x}_w\right)^4 / n}{[\sum f_i w_i \left(x_i - \bar{x}_w\right)^2 / n]^2} - 3$$

Minimum,

$$x_{\min} = \min(x_i)$$

Maximum,

$$x_{\max} = \max(x_i)$$

Example: Summary Statistics

Summary statistics for a small data set are computed.

```
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);

        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 61.7097222222222

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 {33.7826187272065, 194.068533277244}
```

Covariances Class

Summary

Computes the sample variance-covariance or correlation matrix.

`public class Imsl.Stat.Covariances`

Properties

MissingValueMethod

`public int MissingValueMethod {get; set; }`

Description

Sets the method used to exclude missing values in `x` from the computations.

The methods are as follows:

MissingValueMethod	Action
0	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.)
1	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.
2	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.
3	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.

`Double.NaN` is interpreted as the missing value code.

NumRowMissing

```
public int NumRowMissing {get; }
```

Description

Returns the total number of observations that contain any missing values (`Double.NaN`).

Observations

```
public int Observations {get; }
```

Description

Returns the sum of the frequencies.

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.

If `MissingValueMethod` = 0, observations with missing values are not included. Otherwise, all observations are included except for observations with mssing 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.

`System.ArgumentException` id 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.

`Imsl.Stat.TooManyObsDeletedException` id is thrown if more observations have been deleted than were originally entered

i.e. the sum of frequencies has become negative

`Imsl.Stat.MoreObsDelThanEnteredException` id 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` id is thrown if different observations are being deleted than were originally entered

GetIncidenceMatrix

`public int[,] GetIncidenceMatrix()`

Description

Returns the incidence matrix.

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.

Returns

An `int` matrix containing the incidence matrix.

GetMeans

`public double[] GetMeans()`

Description

Returns the means of the variables in `x`.

The components of the array correspond to the columns of `x`.

Returns

A `double` array containing the means of the variables in `x`.

SetFrequencies

`public void SetFrequencies(double[] frequencies)`

Description

The frequency for each observation.

Default: `frequencies[]` = 1.

Parameter

`frequencies` – A `double` array of size `x.GetLength(0)` containing the frequency for each observation.

SetWeights

`public void SetWeights(double[] weights)`

Description

Sets the weight for each observation.

Default: `weights[]` = 1.

Parameter

`weights` – A `double` array of size `x.GetLength(0)` containing the weight for each observation.

Description

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:

$$x_{k0} = 0.0 \quad for\ k = 1, \ldots, p$$

$$c_{jk0} = 0.0 \ \ for\ j, k = 1, \ldots, p$$

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}:

$$r_{i+1} = \frac{f_{i+1} w_{i+1}}{\sum\limits_{l=1}^{i+1} f_l w_l}$$

$$\bar{x}_{k,\, i+1} = \bar{x}_{ki} + \left(x_{k,\, i+1} - \bar{x}_{ki}\right) r_{i+1}$$

$$c_{jk,\, i+1} = c_{jki} + f_{i+1} w_{i+1} \left(x_{j,\, i+1} - \bar{x}_{ji}\right) \left(x_{k,\, i+1} - \bar{x}_{ki}\right) \left(1 - r_{i+1}\right)$$

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.

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.

```
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},
                          {1.0, 4.9, 3.6, 1.4, .1},
                          {1.0, 4.4, 3.0, 1.3, .2},
                          {1.0, 5.1, 3.4, 1.5, .2},
                          {1.0, 5.0, 3.5, 1.3, .3},
                          {1.0, 4.5, 2.3, 1.3, .3},
                          {1.0, 4.4, 3.2, 1.3, .2},
                          {1.0, 5.0, 3.5, 1.6, .6},
                          {1.0, 5.1, 3.8, 1.9, .4},
                          {1.0, 4.8, 3.0, 1.4, .3},
                          {1.0, 5.1, 3.8, 1.6, .2},
```

```
                         {1.0, 4.6, 3.2, 1.4, .2},
                         {1.0, 5.3, 3.7, 1.5, .2},
                         {1.0, 5.0, 3.3, 1.4, .2}};
        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

Summary

Specifies the type of matrix to be computed.

`public enumeration Imsl.Stat.Covariances.MatrixType`

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.

NormOneSample Class

Summary

Computes statistics for mean and variance inferences using a sample from a normal population.

```
public class Imsl.Stat.NormOneSample
```

Properties

ChiSquaredTest

public double ChiSquaredTest {get; }

Description

Returns the test statistic associated with the chi-squared test for variances.

The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where ω_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.

The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where ω_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.

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.

The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where ω_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.

`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.

ConfidenceVariance

```
public double ConfidenceVariance {get; set; }
```

Description

The confidence level (in percent) for two-sided interval estimate of the variances.

`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.

LowerCIVariance

`public double LowerCIVariance {get; }`

Description

Returns the lower confidence limits for the variance.

Mean

`public double Mean {get; }`

Description

Returns the mean of the sample.

StdDev

`public double StdDev {get; }`

Description

Returns the standard deviation of the sample.

TTest

`public double TTest {get; }`

Description

Returns the test statistic associated with the t test.

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.

The t test is a test, against a two-sided alternative, of the null hypothesis value described in `TTestNull`.

TTestNull

`public double TTestNull {get; set; }`

Description

Sets the Null hypothesis value for t test for the mean.

`TTestNull` $= 0.0$ by default.

TTestP

`public double TTestP {get; }`

Description

Returns the probability associated with the t test of a larger t in absolute value.

The t test is a test, against a two-sided alternative, of the null hypothesis value described in `TTestNull`.

UpperCIMean

`public double UpperCIMean {get; }`

Description

Returns the upper confidence limit for the mean.

UpperCIVariance

`public double UpperCIVariance {get; }`

Description

Returns 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.

Parameter

`x` – A one-dimension `double` array containing the observations.

Description

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

$$\bar{x} = \frac{\sum x_i}{n}$$

$$\Delta_s^d Z_t$$

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

$$\chi^2 = \frac{(n-1)\, s^2}{\sigma_0^2}$$

where s is given above. This quantity has a χ^2 distribution with *n - 1* degrees of freedom. Property `ChiSquaredTestDF` returns the degrees of freedom.

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 $H0 : \mu = 20.0$ is tested. The extremely large t value and the correspondingly small p-value provide strong evidence to reject the null hypothesis.

```
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("");

        /* CI variance */
        double ciLoVar = n1samp.LowerCIVariance;
        double ciUpVar = n1samp.UpperCIVariance;
        Console.Out.WriteLine
            ("CI variance is " + ciLoVar + "    " + ciUpVar);
        /*chi-squared test */
        df = n1samp.ChiSquaredTestDF;
        t = n1samp.ChiSquaredTest;
        pvalue = n1samp.ChiSquaredTestP;
        Console.Out.WriteLine("Chi-squared Test results");
        Console.Out.WriteLine("Chi-squared df = " + df);
        Console.Out.WriteLine("Chi-squared t = " + t);
        Console.Out.WriteLine("Chi-squared pvalue = " + pvalue);
    }
}
```

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 = 3.21471738118362E-09

CI variance is 1.33609260499922    6.19986346723949
Chi-squared Test results
Chi-squared df = 14
Chi-squared t = 34.8973333333333
Chi-squared pvalue = 0.0015223176141822
```

NormTwoSample Class

Summary

Computes statistics for mean and variance inferences using samples from two normal populations.

```
public class Imsl.Stat.NormTwoSample
```

Properties

ChiSquaredTest

`public double ChiSquaredTest {get; }`

Description

The test statistic associated with the chi-squared test for common, or pooled, variances.

The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where ω_0^2 is the null hypothesis value as described in `ChiSquaredTestNull`.

ChiSquaredTestDF

`public int ChiSquaredTestDF {get; }`

Description

The degrees of freedom associated with the chi-squared test for the common, or pooled, variances.

The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where ω_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.

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.

The chi-squared test is a test of the hypothesis $\omega^2 = \omega_0^2$ where ω_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.

`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.

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. 272) or Imsl.Stat.NormTwoSample.UpperCICommonVariance (p. 274). Without making the assumption of equal variances, UnequalVariances (p. 274), 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. 274) and UpperCIRatioVariance (p. 275). 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.

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.

FTestDFdenominator

```
public int FTestDFdenominator {get; }
```

Description

The denominator degrees of freedom of the F test for equality of variances.

FTestDFnumerator

```
public int FTestDFnumerator {get; }
```

Description

The numerator degrees of freedom of the F test for equality of variances.

FTestP

```
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.

LowerCICommonVariance

```
public double LowerCICommonVariance {get; }
```

Description

The lower confidence limits for the common, or pooled, variance.

LowerCIDiff

```
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.

If UnequalVariances (p. 274) is `true` then the lower confidence limit for unequal variances will be returned.

LowerCIRatioVariance

`public double LowerCIRatioVariance {get; }`

Description

The approximate lower confidence limit for the ratio of the variance of the first population to the second.

MeanX

`public double MeanX {get; }`

Description

The mean of the first sample, `x`.

MeanY

`public double MeanY {get; }`

Description

The mean of the second sample, `y`.

PooledVariance

`public double PooledVariance {get; }`

Description

The Pooled variance for the two samples.

StdDevX

`public double StdDevX {get; }`

Description

The standard deviation of the first sample, `x`.

StdDevY

`public double StdDevY {get; }`

Description

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.

If UnequalVariances (p. 274) 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.

If UnequalVariances (p. 274) 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.

`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.

If UnequalVariances (p. 274) 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.

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.

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.

If UnequalVariances (p. 274) 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.

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

DowndateX

```
public void DowndateX(double[] x)
```

Description

Removes the observations in `x` from the first sample.

Parameter

`x` – A `double` array containing the values to remove from the first sample.

DowndateY

`public void DowndateY(double[] y)`

Description

Removes the observations in `y` from the second sample.

Parameter

`y` – A `double` array containing the values to remove from the second sample.

Update

`public void Update(double[] x, double[] y)`

Description

Concatenates samples `x` and `y` to the samples provided in the constructor.

Parameters

`x` – A `double` array containing updates to the first sample.

`y` – A `double` array containing updates to the second sample.

UpdateX

`public void UpdateX(double[] x)`

Description

Concatenates the values in `x` to the first sample provided in the constructor.

Parameter

`x` – A `double` array containing updates for the first sample.

UpdateY

`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.

Description

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 μ_1 and σ_1^2 be the mean and variance of the first population, and let μ_2 and σ_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:

$$\bar{x}_1 = \left(\sum x_{1i}/n_1\right), \qquad \bar{x}_2 = \left(\sum x_{2i}\right)/n_2$$

and

$$s_1^2 = \sum \left(x_{1i} - \bar{x}_1\right)^2 / \left(n_1 - 1\right), \qquad s_2^2 = \sum \left(x_{2i} - \bar{x}_2\right)^2 / \left(n_2 - 1\right)$$

Inferences about the Means

The test that the difference in means equals a certain value, for example, μ_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:

$$s^2 = \frac{\left(n_1 - 1\right) s_1 + \left(n_2 - 1\right) s_2}{n_1 + n_2 - 2}$$

The t statistic is as follows:

$$t = \frac{\bar{x}_1 - \bar{x}_2 - \mu_0}{s\sqrt{(1/n_1) + (1/n_2)}}$$

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

$$s_d = \sqrt{(s_1^2/n_1) + (s_2^2/n_2)}$$

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:

$$\mathrm{df} = \frac{s_d^4}{\frac{\left(s_1^2/n_1\right)^2}{n_1-1} + \frac{\left(s_2^2/n_2\right)^2}{n_2-1}}$$

Inferences about Variances

The F statistic for testing the equality of variances is given by $F = s_{\mathrm{max}}^2 / s_{\mathrm{min}}^2$, where s_{max}^2 is the larger of s_1^2 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.

Example 1: NormTwoSample

This example taken from Conover and Iman(1983, p294), involves scores on arithmetic tests of two grade-school classes.

Scores for Standard Group	**Scores for Experimental Group**
72	111
75	118
77	128
80	138
104	140
110	150
125	163
	164
	169

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 ouput. 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.)

```
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 =20.8266655996585
```

Sort Class

Summary

A collection of sorting functions.

`public class Imsl.Stat.Sort`

Constructor

Sort

`public Sort()`

Description

Initializes a new instance of the Imsl.Stat.Sort (p. 280) class.

Methods

Ascending

`static public void Ascending(double[] ra, int[] iperm)`

Description

Sort an array into ascending order.

Parameters

`ra` – A `double` array to be sorted into ascending order.

`iperm` – A `int` array specifying the rearrangement (permutation) of the observations (rows) of `ra`.

Ascending

`static public void Ascending(int[] ra, int[] iperm)`

Description

Sort an array into ascending order.

Parameters

`ra` – an `int`array to be sorted into ascending order

`iperm` – an `int` array to be sorted using the same permutations applied to ra. Typically, you would initialize this to 0, 1, ...

Ascending

`static public void Ascending(double[] ra)`

Description

Sort an array into ascending order.

Parameter

`ra` – A `double` array to be sorted into ascending order.

Ascending

`static public void Ascending(int[] ra)`

Description

Function to sort an integer array into ascending order.

Parameter

`ra` – A `int` array to be sorted into ascending order.

Ascending

`static public void Ascending(double[,] ra, int nKeys)`

Description

Sort a matrix into ascending order by specified keys.

Parameters

ra – A double matrix to be sorted into ascending order.

nKeys – A 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 – A 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 specified keys.

Parameters

ra – A double array to be sorted into ascending order.

nKeys – A int containing the first nKeys columns of ra to be used as the sorting keys.

iperm – A int array specifying the rearrangement (permutation) of the observations (rows) of ra.

Ascending

`static public void Ascending(double[,] ra, int[] indkeys, int[] iperm)`

Description

Sort a matrix into ascending order by specified keys.

Parameters

ra – A double matrix to be sorted into ascending order.

indkeys – A int array containing the order the columns of ra are to be sorted.

iperm – A int array specifying the rearrangement (permutation) of the observations (rows) of ra.

Descending

`static public void Descending(double[] ra, int[] iperm)`

Description

Sort an array into descending order.

Parameters

`ra` – A `double` array to be sorted into descending order.

`iperm` – A `int` array specifying the rearrangement (permutation) of the observations (rows) of `ra`.

Descending

`static public void Descending(double[] ra)`

Description

Sort an array into descending order.

Parameter

`ra` – A `double` array to be sorted into descending order.

Descending

`static public void Descending(double[,] ra, int nKeys)`

Description

Sorts a matrix into descending order by specified keys.

Parameters

`ra` – A `double` matrix to be sorted into descending order.

`nKeys` – A `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

Sorts a matrix into descending order by specified keys.

Parameters

`ra` – A `double` matrix to be sorted into descending order.

`indkeys` – A `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

Sorts an array into descending order by specified keys.

Parameters

`ra` – A `double` array to be sorted into descending order.

`nKeys` – A `int` containing the first `nKeys` columns of `ra` to be used as the sorting keys.

`iperm` – A `int` array specifying the rearrangement (permutation) of the observations (rows) of `ra`.

Descending

`static public void Descending(double[,] ra, int[] indkeys, int[] iperm)`

Description

Sorts a matrix into descending order by specified keys.

Parameters

`ra` – A `double` matrix to be sorted into descending order.

`indkeys` – A `int` array containing the order the columns of `ra` are to be sorted.

`iperm` – A `int` array specifying the rearrangement (permutation) of the observations (rows) of `ra`.

Description

Class `Sort` contains ascending and descending methods for sorting elements of an array or a matrix. The array ascending method sorts the elements of an array, A, into ascending order by algebraic value. The array A is divided into two parts by picking a central element T of the array. The first and last elements of A are compared with T and exchanged until the three values appear in the array in ascending order. The elements of the array are rearranged until all elements greater than or equal to the central element appear in the second part of the array and all those less than or equal to the central element appear in the first part. The upper and lower subscripts of one of the segments are saved, and the process continues iteratively on the other segment. When one segment is finally sorted, the process begins again by retrieving the subscripts of another unsorted portion of the array. On completion, $A_j \leq A_i$ for $j < i$. For more details, see Singleton (1969), Griffin and Redish (1970), and Petro (1970).

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 $i = 0, 1, \ldots, \mathrm{n\_observations} - 2, \mathrm{x}[i][\mathrm{indices\_keys}\ [0]] \leq \mathrm{x}[i+1][\mathrm{indices\_keys}[0]]$
- For $k = 1, \ldots, \mathrm{n\_keys} - 1, \mathrm{if} \mathrm{x}[\mathrm{i}][\mathrm{indices\_keys}[\mathrm{j}]] = \mathrm{x}[\mathrm{i}+1][\mathrm{indices\_keys}[\mathrm{j}]]$ for $j = 0, 1, \ldots, k-1$, then $\mathrm{x}[i][\mathrm{indices\_keys}[k]] = \mathrm{x}[i+1][\mathrm{indices\_keys}[k]]$

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`. The sorting algorithm is based on a quicksort method given by Singleton (1969) with modifications by Griffen and Redish (1970) and Petro (1970).

All other methods in this class work off of the ascending methods.

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.

```
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.

```
using System;
using Imsl.Math;
using Imsl.Stat;
```

```
public 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 = new int[]{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


The Sorted Array - Lowest to Highest

1    1    1
1    1    9
1    1    3
1    1    4
1    1    8
1    2    6
2    1    2
2    2    9
NaN  2    7
2    NaN  5

The permutation array

0
9
2
3
7
5
1
8
6
4
```

Ranks Class

Summary

Compute the ranks, normal scores, or exponential scores for a vector of observations.

`public class Imsl.Stat.Ranks`

Properties

Fuzz

`public double Fuzz {get; set; }`

Description

The fuzz factor used in determining ties.

Random

```
public System.Random Random {get; set; }
```

Description

The `Random` object.

TieBreaker

```
public Imsl.Stat.Ranks.Tie TieBreaker {get; set; }
```

Description

The tie breaker for Ranks.

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).

For tied observations GetNormalScores returns an 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.

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

`x` – 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.

Description

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 $\mathit{TieBreaker} = 0$, $\mathit{Ranks[i]}$ = `GetRanks([j]` = 1.5

if $\mathit{TieBreaker} = 1$, $\mathit{Ranks[i]} = \mathit{Ranks[j]} = 2.0$

if $\mathit{TieBreaker} = 2$, $\mathit{Ranks[i]} = \mathit{Ranks[j]} = 1.0$

if $\mathit{TieBreaker} = 3$, $\mathit{Ranks[i]} = 1.0$ and $\mathit{Ranks[j]} = 2.0$

or *Ranks[i]* = 2.0 and *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, `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 istribution 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 *TieBreaker*, which is set by the property `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 `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 `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.

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();
        ranks.TieBreaker = Imsl.Stat.Ranks.Tie.Highest;
        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();
        ranks.TieBreaker = Imsl.Stat.Ranks.Tie.Lowest;
        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();
        ranks.TieBreaker = Imsl.Stat.Ranks.Tie.Random;
        Imsl.Stat.Random random = new Imsl.Stat.Random(123457);
        random.Multiplier = 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
-0.0414437330939966
 1.60981606718445
 0.775546958322378
 1.17581347255003
 1.36087334286719
 0.0414437330939965
 0.668002132269574
-1.36087334286719
 0.124617407947998
-0.208663745751155
-2.04028132201041
-1.17581347255003
-0.775546958322378
 1.02410618374162
 0.294213138930921
-0.472789120992267
-0.124617407947998
 2.04028132201041
 0.892918486444395
-0.56768639112746
 0.381975767696542
-0.668002132269574
 0.56768639112746
```

```
The Tukey Scores of the Observations - Lowest Score used in Ties

-1.0200762327862
 0.208082136154993
-0.88970115508476
-0.380874057516038
 0.471389465588488
-0.380874057516038
-1.59868725959458
-0.0413298117447387
 1.59868725959458
 0.772935693128221
 1.17060337087942
 1.35372485367826
 0.0413298117447388
 0.665869518001049
-1.35372485367826
 0.124273282084069
```

```
-0.208082136154993
-2.01450973381435
-1.17060337087942
-0.88970115508476
 1.0200762327862
 0.293381232121193
-0.471389465588488
-0.124273282084069
 2.01450973381435
 0.889701155084761
-0.565948821932863
 0.380874057516038
-0.665869518001048
 0.565948821932863


The Van Der Waerden Scores of the Observations - Ties untied by Random

-0.989168627340635
 0.203544231532486
-0.75272879425817
-0.372289360465191
 0.460494539103116
-0.286893916923039
-1.51792915959428
-0.0404405085656462
 1.51792915959428
 0.75272879425817
 1.13097760824516
 1.30015343336342
 0.0404405085656462
 0.649323913186466
-1.30015343336342
 0.121587382750483
-0.203544231532486
-1.84859628850141
-1.13097760824516
-0.864894358685283
 0.989168627340635
 0.286893916923039
-0.460494539103116
-0.121587382750483
 1.84859628850141
 0.864894358685283
-0.552442584646774
 0.372289360465191
-0.649323913186466
 0.552442584646775
```

Ranks.Tie Enumeration

Summary

Determines how to break a tie.

`public enumeration Imsl.Stat.Ranks.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

Summary

Computes empirical quantiles.

`public class Imsl.Stat.EmpiricalQuantiles`

Property

TotalMissing

`public int TotalMissing {get; }`

Description

The total number of missing values.

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.

`Q[i]` corresponds to the empirical quantile at proportion `qProp[i]`. The quantiles are determined by linear interpolation between adjacent ordered sample values.

Returns

A `double` array of length `qProp.Length` containing the empirical quantiles.

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.

Description

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 $1 \le j \le n$; otherwise, the empirical quantile is the smallest or largest order statistic.

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.

```
using System;
using Imsl.Stat;

public 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\t{1}\t\t{2}\t\t{3}",
                qProp[i], XLo[i], Q[i], XHi[i]);
    }
}
```

Output

```
               Smaller        Empirical        Larger
 Quantile       Datum          Quantile         Datum
  0.01 0.32 0.32 0.32
  0.5 1.43 1.47 1.51
  0.9 3 3.081 3.09
  0.95 3.37 3.991 4.75
  0.99 4.75 4.75 4.75
```

TableOneWay Class

Summary

Tallies observations into a one-way frequency table.

`public class Imsl.Stat.TableOneWay`

Properties

Maximum

`public double Maximum {get; }`

Description

The maximum value of `x`.

Minimum

`public double Minimum {get; }`

Description

The minimum value of `x`.

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.

`nIntervals` 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 the closed on the right. Each interval is of length `(max-min)/nIntervals`, where `max` is the maximum value of `x` and `min` is the minimum value of `x`.

Returns

A `double` array containing the one-way frequency table.

GetFrequencyTable

`public double[] GetFrequencyTable(double lowerBound, double upperBound)`

Description

Returns a one-way frequency table using known bounds.

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 `nIntervals - 2` intervals are each of length `(upperBound - lowerBound) / (nIntervals - 2)` and are open on the left and closed on the right. `nIntervals` must be greater than or equal to 3.

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.

GetFrequencyTableUsingClassmarks

```
public double[] GetFrequencyTableUsingClassmarks(double[] classmarks)
```

Description

Returns the one-way frequency table using class marks.

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.

Parameter

classmarks – A double array containing either the cutpoints or the class marks.

Returns

A double array containing the one-way frequency table.

GetFrequencyTableUsingCutpoints

```
public double[] GetFrequencyTableUsingCutpoints(double[] cutpoints)
```

Description

Returns the one-way frequency table using cutpoints.

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.

Parameter

cutpoints – A double array containing the cutpoints.

Returns

A double array containing the one-way frequency table.

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 usings 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.

```
using System;
using Imsl.Stat;

public class TableOneWayEx1
{
    public static void  Main(String[] args)
    {
        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, 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[] 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);
        Console.Out.WriteLine("--------------------------");
        /*  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
0          4
1          8
2          5
3          5
4          3
5          1
6          3
7          0
8          0
9          1
--------------------------
Lower bounds= 0.32
Upper bounds= 4.75
--------------------------
0          2
1          7
2          6
3          6
4          4
5          2
6          2
7          0
8          0
9          1
---------------------
0          2
1          7
2          6
3          6
4          4
5          2
6          2
7          0
8          0
9          1
--------------------
0          2
1          7
2          6
3          6
4          4
5          2
6          2
```

```
7           0
8           0
9           1
```

TableTwoWay Class

Summary

Tallies observations into a two-way frequency table.

`public class Imsl.Stat.TableTwoWay`

Properties

MaximumX

`public double MaximumX {get; }`

Description

The maximum value of x.

MaximumY

`public double MaximumY {get; }`

Description

The maximum value of y.

MinimumX

`public double MinimumX {get; }`

Description

The minimum value of x.

MinimumY

`public double MinimumY {get; }`

Description

The minimum value of 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.

Intervals of equal length are used. Let `xmin` and `xmax` be the minimum and maximum values in x, respectively, with similiar meanings for `ymin` and `ymax`. Then, the first row of the output table is the tally of observations with the x value less than or equal to `xmin + (xmax - xmin)/xIntervals`, and the y value less than or equal to `ymin + (ymax - ymin)/yIntervals`.

Returns

A two-dimensional `double` array containing the two-way frequency table.

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, `xLowerBound, xUpperBound, yLowerBound, yUpperBound`.

The first and last intervals for both variables are semi-infinite in length. `xIntervals` and `yIntervals` must be greater than or equal to 3.

Parameters

`xLowerBound` – A `double` specifies the right endpoint for `x`.

`xUpperBound` – A `double` specifies the left endpoint for `x`.

`yLowerBound` – A `double` specifies the right endpoint for `y`.

`yUpperBound` – A `double` specifies the left endpoint for `y`.

Returns

A two dimensional `double` array containing the two-way frequency table.

GetFrequencyTableUsingClassmarks

```
public double[,] GetFrequencyTableUsingClassmarks(double[] cx, double[] cy)
```

Description

Returns the two-way frequency table using either cutpoints or class marks.

Cutpoints are boundaries and 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.

Parameters

`cx` – A `double` array containing either the cutpoints or the class marks for `x`.

`cy` – A `double` array containing either the cutpoints or the class marks for `y`.

Returns

A two dimensional `double` array containing the two-way frequency table.

GetFrequencyTableUsingCutpoints

```
public double[,] GetFrequencyTableUsingCutpoints(double[] cx, double[] cy)
```

Description

Returns the two-way frequency table using cutpoints.

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.

Parameters

`cx` – A `double` array containing either the cutpoints or the class marks for `x`.

`cy` – A `double` array containing either the cutpoints or the class marks for `y`.

Returns

A two dimensional `double` array containing the two-way frequency table.

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("");
        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("");
        Console.Out.WriteLine("Use Class Marks");
        new PrintMatrix("counts").Print(table);

        double[] cpx = new double[]{1, 2, 3, 4};
        double[] cpy = new double[]{2, 3, 4, 5, 6};
        table = fTbl.GetFrequencyTableUsingCutpoints(cpx, cpy);
        Console.Out.WriteLine("");
        Console.Out.WriteLine("Use Cutpoints");
        new PrintMatrix("counts").Print(table);
    }
}
```

Output

```
Example 1
Use Min and Max for bounds
       counts
   0  1  2  3  4  5
0  4  2  4  2  0  0
1  0  4  3  2  1  0
2  0  0  1  2  0  1
3  0  0  0  0  1  2
4  0  0  0  0  0  1

--------------------------
Lower xbounds= 0.32
Upper xbounds= 4.75
Lower ybounds= 1.47
Upper ybounds= 6.37
--------------------------

Use Known bounds
       counts
   0  1  2  3  4  5
0  3  2  4  0  0  0
```

```
1  0  5  5  2  0  0
2  0  0  1  3  2  0
3  0  0  0  0  0  2
4  0  0  0  0  1  0


Use Class Marks
       counts
   0  1  2  3  4  5
0  3  2  4  0  0  0
1  0  5  5  2  0  0
2  0  0  1  3  2  0
3  0  0  0  0  0  2
4  0  0  0  0  1  0


Use Cutpoints
       counts
   0  1  2  3  4  5
0  3  2  4  0  0  0
1  0  5  5  2  0  0
2  0  0  1  3  2  0
3  0  0  0  0  0  2
4  0  0  0  0  1  0
```

TableMultiWay Class

Summary

Tallies observations into a multi-way frequency table.

`public class Imsl.Stat.TableMultiWay`

Properties

BalancedTable

`public Imsl.Stat.TableMultiWay.TableBalanced BalancedTable {get; }`

Description

An object containing the balanced table.

UnbalancedTable

`public Imsl.Stat.TableMultiWay.TableUnbalanced UnbalancedTable {get; }`

Description

An object containing the unbalanced table.

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.

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`.

Returns

A `int` array containing the number of observations (row) in each group.

SetFrequencies

`public void SetFrequencies(double[] frequencies)`

Description

Sets the frequencies for each observation in x.

Length of input must be the same as the number of observations or number of rows in `x`.

Default `frequencies`[] = 1.

Parameter

`frequencies` – A `double` array containing the frequency for each observation in x.

Description

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.

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},
                {4.5, 5.5}, {2.5, 4.5},
                {0.5, 3.5}, {1.5, 2.5},
                {0.5, 2.5}, {2.5, 5.5}};

        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("");
        Console.Out.WriteLine("");
        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("");
        Console.Out.WriteLine("");
        Console.Out.WriteLine("        Table");



        Console.Out.Write("      ");
        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
3.5   0   0   0   0   0   2
4.5   0   0   0   0   1   0
```

Example 3: TableMultiWay

The unbalanced table of frequencies for a data matrix of size 4 x 3 is output.

```
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();
        mf.SetNoColumnLabels();
        //  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
2
5
2
6


Unbalanced Table

2
3
1
4
```

TableMultiWay.TableBalanced Class

Summary

Tallies the number of unique values of each variable.

`public class Imsl.Stat.TableMultiWay.TableBalanced`

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.

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.

Returns

A `double` array containing the frequencies for each variable in `x`.

GetValues

`public double[] GetValues()`

Description

Returns the values of the classification variables.

`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`.

Returns

A `double` array containing the values of the classification variables.

TableMultiWay.TableUnbalanced Class

Summary

Tallies the frequency of each cell in `x`.

`public class Imsl.Stat.TableMultiWay.TableUnbalanced`

Property

NCells

`public int NCells {get; }`

Description

Returns the number of non-empty cells.

Methods

GetListCells

`public double[] GetListCells()`

Description

Returns for each row, a list of the levels of `nkeys` coorresponding 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

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

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_1 \;\; i = 1, 2, \ldots, n$$

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, β_0 and β_1 are the intercept and slope parameters (respectively) and the ε_1's are independently distributed normal errors, each with mean 0 and variance σ^2.

The multiple linear regression model is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_k x_{ik} + \varepsilon_1 \ i = 1, 2, \ldots, n$$

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 k independent (explanatory) variables; $\beta_0, \beta_1, \ldots, \beta_k$ are the regression coefficients; and the ε_1's are independently distributed normal errors, each with mean 0 and variance σ^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 β_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 $(x_1, x_2, \ldots, x_k, y)^T (x_1, x_2, \ldots, x_k, y)$.

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

$$y_i = f(x_i; \theta) + \varepsilon_i \; i = 1, 2, \ldots, n$$

where the observed values of the y_i's constitute the responses or values of the dependent variable, the x_i's are the known vectors of values of the independent (explanatory) variables, f is a known function of an unknown regression parameter vector θ, and the ε_i's are independently distributed normal errors each with mean 0 and variance σ^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 ε_i is assumed to equal σ^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 $y = X\beta + \varepsilon$, 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 $\varepsilon$is the $n \times 1$ vector of errors whose elements are independently normally distributed with mean 0 and variance σ^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 `ANOVA.GetArray()` method. This returns a one-dimensional array. In `StepwiseRegression`, `ANOVA.GetArray()` returns `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 `ANOVA.GetArray()` and the notation frequently used for these is described in the following table (here, `AOV = ANOVA.GetArray()`):

Model Analysis of Variance Table

Variation Src.	**Deg. of Freedom**	**Sum of Squares**	**Mean Square**	F	p-**value**
Regression	DFR = AOV[0]	SSR = AOV[3]	MSR = AOV[6]	`AOV[8]`	`AOV[9]`
Error	DFE = AOV[1]	SSE = AOV[4]	s^2 = AOV[7]		
Total	DFT = AOV[2]	SST = AOV[5]			

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:

$$\mathrm{SST} = \sum_{i=1}^{n} w_i \left(y_i - \bar{y}\right)^2$$

If the model does not have an intercept (`hasIntercept = 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:

$$\mathrm{SST} = \sum_{i=1}^{n} w_i y_i^2$$

The error sum of squares is given as follows:

$$\mathrm{SSE} = \sum_{i=1}^{n} w_i \left(y_i - \hat{y}_i\right)^2$$

The error degrees of freedom is defined by $\mathrm{DFE} = n - r$.

The estimate of σ^2 is given by $s^2 = \mathrm{SSE/DFE}$, 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 $F = s^2 = \mathrm{MSR}/s^2$. The p-value associated with the test is the probability of an F larger than that computed 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

$$R_a^2 = (\text{AOV}[11])$$

are expressed as a percentage and are defined as follows:

$$R^2 = 100\,(\text{SSR}/\text{SST}) = 100\,(1 - \text{SSE}/\text{SST})$$

$$R_a^2 = 100 \max\left\{0, 1 - \frac{s^2}{\text{SST}/\text{DFT}}\right\}$$

The square root of s^2 ($s = \text{AOV}[12]$) 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 ($CV = \text{AOV}[14]$) is expressed as a percentage and defined by $\text{CV} = 100s/\bar{y}$.

`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 rank(X) $= r$, β is the $p \times 1$ vector of regression coefficients, and ε is the $n \times 1$ vector of errors whose elements are independently normally distributed with mean 0 and variance ϕ^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

$$h_i = [x_i^T \left(X^T W X\right)^- x_i] w_i$$

where $W = \text{diag}\,(w_1, w_2 \ldots, w_n)$ and $\left(X^T W T\right)^-$ 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 $(1 - h_i)s^2 w_i$, 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

$$r_i = e_i \sqrt{\frac{w_i}{s^2\left(1 - h_i\right)}}$$

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 $e_i / \left(1 - h_i\right)$. 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:

$$s_i^2 = \frac{\left(n - r\right)s^2 - w_i e_i^2 / \left(1 - h_i\right)}{n - r - 1}$$

The jackknife residual is defined as

$$t_i = e_i \sqrt{\frac{w_i}{s_i^2\left(1 - h_i\right)}}$$

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:

$$D_i = \frac{w_i h_i e_i^2}{r s^2 \left(1 - h_i\right)^2}$$

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.

$$\mathrm{DFFITS}_i = e_i \sqrt{\frac{w_i h_i}{s_i^2 \left(1 - h_i\right)^2}}$$

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

$$\left(x_1, x_2, x_1^2, x_2^2, x_1 x_2\right)$$

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

$$y = e^{\beta_0 + \beta_1 x_1} \varepsilon$$

can be transformed to a model that satisfies the linear regression model provided the ε_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.

LinearRegression Class

Summary

Fits a multiple linear regression model with or without an intercept.

```
public class Imsl.Stat.LinearRegression
```

Properties

ANOVA

```
public Imsl.Stat.ANOVA ANOVA {get; }
```

Description

Returns an analysis of variance table and related statistics.

CoefficientTTests

```
public Imsl.Stat.LinearRegression.CoefficientTTestsValue CoefficientTTests
{get; }
```

Description

Returns statistics relating to the regression coefficients.

HasIntercept

```
public bool HasIntercept {get; }
```

Description

A `bool` which indicates whether or not an intercept is in this regression model.

Rank

```
public int Rank {get; }
```

Description

Returns the rank of the matrix.

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

```
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.

GetCaseStatistics

```
virtual public Imsl.Stat.LinearRegression.CaseStatistics
  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)
```

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

```
virtual public Imsl.Stat.LinearRegression.CaseStatistics
  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.

GetCoefficients

```
public double[] GetCoefficients()
```

Description

Returns the regression coefficients.

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.

Returns

A `double` array containing the regression coefficients.

`Imsl.Math.SingularMatrixException` id is thrown when the regression matrix is singular

GetR

`public double[,] GetR()`

Description

Returns a copy of the R matrix.

R is the upper triangular matrix containing the R matrix from a QR decomposition of the matrix of regressors.

Returns

A `double` matrix containing a copy of the R matrix.

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.

`x.length` must be equal to the number of variables set in the constructor.

Parameters

`x` – A `double` array containing the independent (explanatory) variables.

`y` – A `double` representing the dependent (response) variable.

Update

`public void Update(double[] x, double y, double w)`

Description

Updates the regression object with a new observation and weight.

`x.length` must be equal to the number of variables set in the constructor.

Parameters

`x` – A `double` array containing the independent (explanatory) variables.

`y` – A `double` representing the dependent (response) variable.

`w` – A `double` representing the weight.

Update

```
public void Update(double[,] x, double[] y)
```

Description

Updates the regression object with a new set of observations.

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.

Parameters

`x` – A `double` matrix containing the independent (explanatory) variables.

`y` – A `double` array containing the dependent (response) variables.

Update

```
public void Update(double[,] x, double[] y, double[] w)
```

Description

Updates the regression object with a new set of observations and weights.

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.

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.

Description

Fits a multiple linear regression model with or without an intercept. If the constructor argument `hasIntercept` is true, the multiple linear regression model is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ \ldots + \beta_k x_{ik} + \varepsilon_i \quad i = 1,\, 2,\, \ldots,\, n$$

where the observed values of the y_i's constitute the responses or values of the dependent variable, the x_{i1}'s, x_{i2}'s, $\ldots, x_{ik}$'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 σ^2/w_i. If `hasIntercept` is `false`, β_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. 335) 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 $W = diag(w_1, w_2, ..., w_n)$. The leverage is defined as

$$h_i = [x_i^T (X^T W X)^- x_i] w_i$$

(In the case of linear equality restrictions on β, the leverage is defined in terms of the reduced model.) Put $D = diag(d_1, d_2, ..., d_k)$ with $d_j = 1$ if the j-th diagonal element of R is positive and 0 otherwise. The leverage is computed as $h_i = (a^T D a) w_i$ 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 $h_i s^2 / w_i$, where $s^2 = SSE/DFE$. 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 ith 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 ith 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 ith jackknife residual or deleted residual involves the difference between y_i and its predicted value based on the data set in which the ith 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 ith 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 ith case is a measure of how much an individual case affects the estimated regression coefficients. It is given by

$$D_i = \frac{w_i h_i e_i^2}{r s^2 (1 - h_i)^2}$$

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 ith 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.

Example: Linear Regression

The coefficients of a simple linear regression model, without an intercept, are computed.

```
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] + "}");
    }
}
```

Output

```
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.

```
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, 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]);
            cs.Effects = -2;
            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);
        p.Print(mf, results);
    }
}
```

Output

```
                                        Selected Case Statistics
   Jackknife Residual.        Cook's D              DFFITS          [Conf. Interval     on the Mean]
0    -0.343038692852844  0.0448855192564415    -0.323965838130963  2.26094652131247   3.99619633583039
1    -0.327326835353989  0.0183908045977011    -0.207019667802706  3.4674120686278    4.81830221708648
2    -0.337597012047161  0.0111298613543336    -0.161225169613381  4.6125816288173    5.70170408546842
3  Infinity              0.275862068965519   Infinity              5.64823106667496   6.69462607618219
4    -0.417763902391217  0.0235122737669971    -0.236601469253295  6.5629846966826    7.80844387474598
5     NaN                NaN                    NaN                6.73635797432486   9.26364202567514
6    -0.742307488958098  0.372413793103455     -0.995910003310489  8.2011181029417   10.2274533256297
```

LinearRegression.CaseStatistics Class

Summary

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.

```
public class Imsl.Stat.LinearRegression.CaseStatistics
```

Properties

ConfidenceInterval

```
virtual public double[] ConfidenceInterval {get; }
```

Description

Returns the Confidence Interval on the mean for an observation.

ConLevelMean

```
virtual public double ConLevelMean {set; }
```

Description

Sets the confidence level for two-sided interval estimates on the mean, in percent.

Default = 0.95.

ConLevelPred

```
virtual public double ConLevelPred {set; }
```

Description

Sets the confidence level for two-sided prediction intervals, in percent.

Default = 0.95.

CooksDistance

```
virtual public double CooksDistance {get; }
```

Description

Returns Cook's Distance for an observation.

DFFITS

```
virtual public double DFFITS {get; }
```

Description

Returns `DFFITS` for an observation.

Effects

`virtual public int Effects {set; }`

Description

Sets the effect option.

The absolute value is used to specify the number of effects (sources of variation) due to the model. The sign of `Effect` specifies the following:

Effects	*Meaning*
< 0	Each effect corresponds to a single regressor (coefficient) in the model.
> 0	Currently not used. This will result in an `IllegalArgumentException` being thrown.
0	There are no effects in the model. `hasIntercept` must be set to `true`.

Default = -1.

JackknifeResidual

`virtual public double JackknifeResidual {get; }`

Description

Returns the Jackknife Residual for an observation.

Leverage

`virtual public double Leverage {get; }`

Description

Returns the `Leverage` for an observation.

ObservedResponse

`virtual public double ObservedResponse {get; }`

Description

Returns the observed response for an observation.

PredictedResponse

`virtual public double PredictedResponse {get; }`

Description

Returns the predicted response for an observation.

PredictionInterval

```
virtual public double[] PredictionInterval {get; }
```

Description

Returns the Prediction Interval for an observation.

Residual

```
virtual public double Residual {get; }
```

Description

Returns the `Residual` for an observation.

StandardizedResidual

```
virtual public double StandardizedResidual {get; }
```

Description

Returns the Standardized Residual for an observation.

Statistics

```
virtual public double[] Statistics {get; }
```

Description

Returns the case statistics for an observation.

Elements 0 through 11 contain the following:

Index	*Description*
0	Observed response
1	Predicted response
2	Residual
3	Leverage
4	Standardized residual
5	Jackknife residual
6	Cook's distance
7	DFFITS
8,9	Confidence interval on the mean
10,11	Prediction interval

Example: Linear Regression Case Statistics

Selected case statistics of a simple linear regression model, with an intercept, are computed.

```
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, 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]);
            cs.Effects = -2;
            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);
        p.Print(mf, results);
    }
}
```

Output

```
                                        Selected Case Statistics
   Jackknife Residual.        Cook's D             DFFITS          [Conf. Interval     on the Mean]
0    -0.343038692852844  0.0448855192564415    -0.323965838130963  2.26094652131247   3.99619633583039
1    -0.327326835353989  0.0183908045977011    -0.207019667802706  3.4674120686278    4.81830221708648
2    -0.337597012047161  0.0111298613543336    -0.161225169613381  4.6125816288173    5.70170408546842
3  Infinity              0.275862068965519   Infinity              5.64823106667496   6.69462607618219
4    -0.417763902391217  0.0235122737669971    -0.236601469253295  6.5629846966826    7.80844387474598
5     NaN                NaN                    NaN                6.73635797432486   9.26364202567514
6    -0.742307488958098  0.372413793103455     -0.995910003310489  8.2011181029417   10.2274533256297
```

LinearRegression.CoefficientTTestsValue Class

Summary

`CoefficientTTestsValue` contains statistics related to the regression coefficients.

`public class Imsl.Stat.LinearRegression.CoefficientTTestsValue`

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

Summary

Fits a multivariate nonlinear regression model using least squares.

`public class Imsl.Stat.NonlinearRegression`

Properties

AbsoluteTolerance

`virtual public double AbsoluteTolerance {set; }`

Description

The absolute function tolerance.

The tolerance must be greater than or equal to zero.

The default value is 4.93e-32.

Coefficients

`virtual public double[] Coefficients {get; }`

Description

The regression coefficients.

DFError

`virtual public double DFError {get; }`

Description

The degrees of freedom for error.

Digits

`virtual public int Digits {set; }`

Description

The number of good digits in the residuals.

The number of digits must be greater than zero.

The default value is 15.

ErrorStatus

`virtual public int ErrorStatus {get; }`

Description

Characterizes the performance of `NonlinearRegression`.

Value	Description
0	All convergence tests were met.
1	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.
2	Scaled actual and predicted reductions in the function are less than or equal to the relative function convergence tolerance `RelativeTolerance`.
3	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.
4	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.

See Also: RelativeTolerance (p. 343), StepTolerance (p. 343), MaxStepsize (p. 342)

FalseConvergenceTolerance

`virtual public double FalseConvergenceTolerance {set; }`

Description

The false convergence tolerance.

The tolerance must be greater than or equal to zero.

The default value is 2.22e-14.

GradientTolerance

`virtual public double GradientTolerance {set; }`

Description

The gradient tolerance.

The tolerance must be greater than or equal to zero.

The default value is 6.055e-6.

Guess

`virtual public double[] Guess {set; }`

Description

The initial guess of the parameter values.

The default value is an array of zeroes.

InitialTrustRegion

`virtual public double InitialTrustRegion {set; }`

Description

The initial trust region radius.

The initial trust radius must be greater than zero.

The default value is set based on the initial scaled Cauchy step.

MaxIterations

`virtual public int MaxIterations {set; }`

Description

The maximum number of iterations allowed during optimization

The value must be greater than 0.

The default value is 100.

MaxStepsize

`virtual public double MaxStepsize {set; }`

Description

The maximum allowable stepsize.

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*.

R

`virtual public double[,] R {get; }`

Description

A copy of the R matrix.

The upper triangular matrix containing the R matrix from a QR decomposition of the matrix of regressors.

Rank

`virtual public int Rank {get; }`

Description

The rank of the matrix.

RelativeTolerance

`virtual public double RelativeTolerance {set; }`

Description

The relative function tolerance

The relative function tolerance must be greater than or equal to zero.

The default value is 1.0e-20.

Scale

`virtual public double[] Scale {set; }`

Description

The scaling array for *theta*.

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. The default value is an array of ones.

StepTolerance

`virtual public double StepTolerance {set; }`

Description

The step tolerance.

The step tolerance must be greater than or equal to zero.

The default value is 3.667e-11.

Constructor

NonlinearRegression

`public NonlinearRegression(int nparm)`

Description

Constructs a new nonlinear regression object.

Parameter

`nparm` – An `int` which specifies the number of unknown parameters in the regression.

Methods

GetCoefficient

`virtual public double GetCoefficient(int i)`

Description

Returns the estimate for a coefficient.

Parameter

`i` – An `int` which specifies the index of a coefficient whose estimate is to be returned.

Returns

A `double` which contains the estimate for the i-th coefficient or `null` if Imsl.Stat.NonlinearRegression.Solve(Imsl.Stat.NonlinearRegression.IFunction) (p. 344) has not been called.

GetSSE

`virtual public double GetSSE()`

Description

Returns the sums of squares for error.

Returns

A `double` which contains the sum of squares for error or `null` if Imsl.Stat.NonlinearRegression.Solve(Imsl.Stat.NonlinearRegression.IFunction) (p. 344) has not been called.

Solve

```
virtual public double[] Solve(Imsl.Stat.NonlinearRegression.IFunction F)
```

Description

Solves the least squares problem and returns the regression coefficients.

Parameter

`F` – An Imsl.Stat.NonlinearRegression.IFunction (p. 354) whose coefficients are to be computed.

Returns

A `double` array containing the regression coefficients.

`Imsl.Stat.TooManyIterationsException` id is thrown when the number of allowed iterations is exceeded

`Imsl.Stat.NegativeFreqException` id is thrown when the specified frequency is negative

`Imsl.Stat.NegativeWeightException` id is thrown when the weight is negative

Description

The nonlinear regression model is

$$y_i = f(x_i; \theta) + \varepsilon_i \qquad i = 1, 2, \ldots, n$$

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, θ is the vector of p regression parameters, and the ε_i are independently distributed normal errors each with mean zero and variance σ^2. For this model, a least squares estimate of θ is also a maximum likelihood estimate of θ.

The residuals for the model are

$$e_i(\theta) = y_i - f(x_i; \theta) \qquad i = 1, 2, \ldots, n$$

A value of θ that minimizes

$$\sum_{i=1}^{n} [e_i(\theta)]^2$$

is the least-squares estimate of θ 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 θ. 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 $||s_c||_2 < \delta_c$, this update is accepted; otherwise, μ_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 x 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 ϵ_i in the nonlinear regression model is not constant but instead is σ^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 $\hat{\beta}_0$ and $\hat{\beta}_1$ from a simple linear regression of *ln y* on *ln x*. The starting values for the nonlinear regression in this case would be

   $$\theta_1 = e^{\hat{\beta}_0}\, and\, \theta_2 = \hat{\beta}_1$$

   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 θ 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. 342) 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. 341) to indicate the actual number of good digits in the residuals.

3. The model is discontinuous as a function of θ. There may be a mistake in the user-supplied function. Note that the function $f(x;\theta)$ can be a discontinuous function of x.

4. The R matrix value given by Imsl.Stat.NonlinearRegression.R (p. 343) is inaccurate. If only a function is supplied try providing the Imsl.Stat.NonlinearRegression.IDerivative (p. 353). If the derivative is supplied try providing only Imsl.Stat.NonlinearRegression.IFunction (p. 354).

5. Overflow occurs during the computations. Make sure the user-supplied functions do not overflow at some value of θ.

6. The estimate of θ is going to infinity. A parameterization of the problem in terms of reciprocals may help.

7. Some components of θ 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. 344) method must be called before using any property as a right operand, otherwise the value is `null`.

Example 1: Nonlinear Regression using Finite Differences

In this example a nonlinear model is fitted. The derivatives are obtained by finite differences.

```
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;
                e[0] = ydata[iobs] - theta[0] * Math.Exp(theta[1] * xdata[iobs]);
            }
            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 {58.6065629385189, -0.0395864472964795}
The computed rank is 2
The degrees of freedom for error are 13
The sums of squares for error is 49.4592998624719
     R from the QR decomposition
          0                 1
0  1.87385998095046  1139.92835934133
1  0                 1139.79755002385
```

Example 2: Nonlinear Regression with User-supplied Derivatives

In this example a nonlinear model is fitted. The derivatives are supplied by the user.

```
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;
                e[0] = ydata[iobs] - theta[0] * Math.Exp(theta[1] * xdata[iobs]);
            }
            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;
                de[0] = - Math.Exp(theta[1] * xdata[iobs]);
                de[1] = (- theta[0]) * xdata[iobs] *
                    Math.Exp(theta[1] * xdata[iobs]);
            }
            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  1139.92837730064
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 recipricol of the starting value. The derivatives are obtained by finite differences.

```
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;
                e[0] = ydata[iobs] - theta[0] * Math.Exp(theta[1] * xdata[iobs]);
```

```
            }
            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 {5.78378362045064E-09, -0.0396252538454606}
The computed rank is 2
The degrees of freedom for error are 13
The sums of squares for error is 5.1663766194061E-19
       R from the QR decomposition
          0                   1
0  1.87310563181707  5.74734586570442E-09
1  0                 5.83713991871454E-11
```

NonlinearRegression.IDerivative Interface

Summary

Public interface for the user supplied function to compute the derivative for `NonlinearRegression`.

```
public interface Imsl.Stat.NonlinearRegression.IDerivative :
Imsl.Stat.NonlinearRegression.IFunction
```

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.

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.

Parameters

`theta` – An input `double` array which contains the parameter values of the regression function.

`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 the observation `y[iobs]`.

`de` – An output `double` array containing the partial derivatives of the error (residual) for observation `y[iobs]`.

Returns

A `boolean` 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.

NonlinearRegression.IFunction Interface

Summary

Public interface for the user supplied function for `NonlinearRegression`.

`public interface Imsl.Stat.NonlinearRegression.IFunction`

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.

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).

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 `boolean` 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.

SelectionRegression Class

Summary

Selects the best multiple linear regression models.

`public class Imsl.Stat.SelectionRegression`

Properties

CriterionOption

`virtual public Imsl.Stat.SelectionRegression.Criterion CriterionOption {get; set; }`

Description

The criterion option used to calculate the regression estimates.

By default for all criteria, subset size 1,2, ..., $k = nCandidate$ are considered. However, for R^2 the maximum number of subsets can be restricted using property Imsl.Stat.SelectionRegression.MaximumSubsetSize (p. 356).

Criterion Option	Description
RSquared	For R^2, subset sizes 1, 2, ..., `MaximumSubsetSize` are examined. This is the default with `MaximumSubsetSize` $= nCandidate$.
AdjustedRSquared	For Adjusted R^2, subset sizes 1, 2, ..., $nCandidate$ are examined.
MallowsCP	For Mallow's C_p Subset sizes 1, 2, ..., $nCandidate$ are examined.

See Also: RSquared (p. 366), AdjustedRSquared (p. 366), MallowsCP (p. 366)

MaximumBestFound

`virtual public int MaximumBestFound {set; }`

Description

The maximum number of best regressions to be found.

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. The default value is 1.

See Also: RSquared (p. 366), AdjustedRSquared (p. 366), MallowsCP (p. 366)

MaximumGoodSaved

`virtual public int MaximumGoodSaved {set; }`

Description

The maximum number of good regressions for each subset size saved.

`MaximumGoodSaved` must be greater than or equal to Imsl.Stat.SelectionRegression.MaximumBestFound (p. 355). 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. 356)).

MaximumSubsetSize

`virtual public int MaximumSubsetSize {set; }`

Description

The maximum subset size if R^2 criterion is used.

Default: `MaximumSubsetSize` = *nCandidate*.

See Also: RSquared (p. 366), AdjustedRSquared (p. 366), MallowsCP (p. 366)

Statistics

`virtual public Imsl.Stat.SelectionRegression.SummaryStatistics Statistics {get; }`

Description

A `SummaryStatistics` object.

Constructor

SelectionRegression

`public SelectionRegression(int nCandidate)`

Description

Constructs a new `SelectionRegression` object.

nCandidate must be greater than 2.

Parameter

`nCandidate` – An `int` containing the number of candidate variables (independent variables).

Methods

Compute

`virtual public void Compute(double[,] cov, int nObservations)`

Description

Computes the best multiple linear regression models using a user-supplied covariance matrix.

cov can be computed using the Imsl.Stat.Covariances (p. 257) class.

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*.

`Imsl.Stat.NoVariablesException` id is thrown if no variables can enter any model

Compute

```
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.

The number of columns in x must be equal to the number of variables set in the constructor.

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.

`Imsl.Stat.NoVariablesException` id is thrown if no variables can enter any model

`Imsl.Stat.NegativeFreqException` id is thrown if a frequency is less than zero.

`Imsl.Stat.NegativeWeightException` id is thrown if a weight is less than zero.

`Imsl.Stat.TooManyObsDeletedException` id is thrown if more observations have been deleted than were originally entered

`Imsl.Stat.MoreObsDelThanEnteredException` id is thrown if more observations are being deleted from the output covariance matrix than were originally entered

`Imsl.Stat.DiffObsDeletedException` id 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.

The number of columns in x must be equal to the number of variables set in the constructor.

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.

`Imsl.Stat.NoVariablesException` id is thrown if no variables can enter any model

`Imsl.Stat.NegativeWeightException` id is thrown if a weight is less than zero.

`Imsl.Stat.TooManyObsDeletedException` id is thrown if more observations have been deleted than were originally entered

`Imsl.Stat.MoreObsDelThanEnteredException` id is thrown if more observations are being deleted from the output covariance matrix than were originally entered

`Imsl.Stat.DiffObsDeletedException` id 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)
```

Description

Computes the best multiple linear regression models.

The number of columns in x must be equal to the number of variables set in the constructor.

Parameters

`x` – A `double` matrix containing the observations of the candidate (independent) variables.

`y` – A `double` array containing the observations of the dependent variable.

`Imsl.Stat.NoVariablesException` id is thrown if no variables can enter any model

`Imsl.Stat.TooManyObsDeletedException` id is thrown if more observations have been deleted than were originally entered

`Imsl.Stat.MoreObsDelThanEnteredException` id is thrown if more observations are being deleted from the output covariance matrix than were originally entered

`Imsl.Stat.DiffObsDeletedException` id is thrown if different observations are being deleted from the return matrix than were originally entered

Description

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[0:,0:],System.Double[]) (p. 358) method.

"Best" is defined by using one of the following three criteria:

- R^2 (in percent)

$$R^2 = 100(1 - \frac{\text{SSE}_p}{\text{SST}})$$

- R_a^2 (adjusted R^2)

$$R_a^2 = 100[1 - (\frac{n-1}{n-p})\frac{\text{SSE}_p}{\text{SST}}]$$

  Note that maximizing the R_a^2 is equivalent to minimizing the residual mean squared error:

$$\frac{\text{SSE}_p}{(n-p)}$$

- Mallow's C_p statistic

$$C_p = \frac{\text{SSE}_p}{s_{\text{k}}^2} + 2p - n$$

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 β_0 (or p - 1 of the k candidate variables). Variable

$$S_{\text{k}}^2$$

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 *nObservations* 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 *cov*. In this case, `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 *cov* are the same as in the previous case. Argument *nObservations* 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 *nObservations* 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 $\mathrm{k} + 1 > -\log_2(\epsilon)$, where ϵ 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 $\mathrm{k} > 49$) 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.

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.

```
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,
                                        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.Compute(x, y);
        SelectionRegression.SummaryStatistics stats = sr.Statistics;

        for (int i = 1; i <= 4; i++)
        {
            double[] tmpCrit = stats.GetCriterionValues(i);
            int[,] indvar = stats.GetIndependentVariables(i);
            Console.Out.WriteLine("Regressions with "+i+" variable(s)  (R-squared)");
            for (int j = 0; j < tmpCrit.GetLength(0); j++)
            {
                Console.Out.Write("     " + tmpCrit[j] + "        ");
```

```
                for (int k = 0; k < indvar.GetLength(1); k++)
                    Console.Out.Write(indvar[j,k] + "   ");
                Console.Out.WriteLine("");
            }
            Console.Out.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.Out.WriteLine("\n\nRegressions with "+(i+1)+" variable(s)  (R-squared)");
            Console.Out.WriteLine("Variable   Coefficient   Standard Error  t-statistic   p-value");
            pm.Print(tst, tmpCoef);
        }
    }
}
```

Output

```
Regressions with 1 variable(s)  (R-squared)
     67.4541964131609        4
     66.6268257633294        2
     53.3948023835034        1
     28.5872731229812        3

Regressions with 2 variable(s)  (R-squared)
     97.8678374535632        1   2
     97.2471047716931        1   4
     93.5289640615808        3   4
     68.006040795005        2   4
     54.8166748844824        1   3

Regressions with 3 variable(s)  (R-squared)
     98.2335451200427        1   2   4
     98.2284679219087        1   2   3
     98.1281092587344        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).

```
using System;
using Imsl.Math;
using Imsl.Stat;

public 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.CriterionOption = Imsl.Stat.SelectionRegression.Criterion.MallowsCP;
        sr.MaximumBestFound = 3;
        sr.Compute(x, y);
        SelectionRegression.SummaryStatistics stats = sr.Statistics;

        for (int i = 1; i <= 4; i++)
        {
            double[] tmpCrit = stats.GetCriterionValues(i);
            int[,] indvar = stats.GetIndependentVariables(i);
            Console.Out.WriteLine("Regressions with "+i+" variable(s)  (MallowsCP)");
            for (int j = 0; j < tmpCrit.GetLength(0); j++)
            {
                Console.Out.Write("     " + tmpCrit[j] + "        ");
                for (int k = 0; k < indvar.GetLength(1); k++)
                    Console.Out.Write(indvar[j,k] + "   ");
                Console.Out.WriteLine("");
            }
            Console.Out.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[,] tmpCoef = stats.GetCoefficientStatistics(i);
            Console.Out.WriteLine("\n\nRegressions with "+(i+1)+" variable(s)  (MallowsCP)");
            Console.Out.WriteLine("Variable   Coefficient   Standard Error  t-statistic   p-value");
            pm.Print(tst, tmpCoef);
        }
    }
}
```

Output

```
Regressions with 1 variable(s)  (MallowsCP)
     138.730833491679        4
     142.486406936963        2
     202.548769123452        1
     315.154284140084        3

Regressions with 2 variable(s)  (MallowsCP)
     2.67824159831843        1   2
     5.49585082475865        1   4
     22.3731119646976        3   4
     138.225919754643        2   4
     198.094652569591        1   3

Regressions with 3 variable(s)  (MallowsCP)
     3.01823347348735        1   2   4
     3.04127972306417        1   2   3
     3.49682444234848        1   3   4
     7.33747399565598        2   3   4

Regressions with 4 variable(s)  (MallowsCP)
     5        1   2   3   4



Regressions with 1 variable(s)  (MallowsCP)
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 2 variable(s)  (MallowsCP)
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 3 variable(s)  (MallowsCP)
Variable   Coefficient   Standard Error  t-statistic   p-value

1.000         1.696         0.205         8.290          0.000
2.000         0.657         0.044         14.851         0.000
3.000         0.250         0.185         1.354          0.209
```

SelectionRegression.Criterion Enumeration

Summary

Criterion Methods.

`public enumeration Imsl.Stat.SelectionRegression.Criterion`

Fields

`AdjustedRSquared`
`public Imsl.Stat.SelectionRegression.Criterion AdjustedRSquared`

Description

Indicates R_a^2 (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

Summary

`SummaryStatistics` contains statistics related to the regression coefficients.

`public class Imsl.Stat.SelectionRegression.SummaryStatistics`

Methods

GetCoefficientStatistics
`virtual public double[,] GetCoefficientStatistics(int regressionIndex)`

Description

Returns the coefficients statistics for each of the best regressions found for each subset considered.

The value set using Imsl.Stat.SelectionRegression.MaximumBestFound (p. 355) determines the total number of best regressions to find. The number of best regression is equal to (Imsl.Stat.SelectionRegression.MaximumSubsetSize (p. 356) 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):

Column	Description
0	variable number
1	coefficient estimate
2	estimated standard error of the estimate
3	t-statistic for the test that the coefficient is 0
4	p-value for the two-sided t test

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. 366), AdjustedRSquared (p. 366), MallowsCP (p. 366)

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.

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 Imsl.Stat.SelectionRegression.MaximumSubsetSize (p. 356) rows and *nCandidate* columns containing the criterion values.

GetIndependentVariables

```
virtual public int[,] GetIndependentVariables(int numVariables)
```

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 Imsl.Stat.SelectionRegression.SummaryStatistics.GetCriterionValues(System.Int32) (p. 367).

Parameter

`numVariables` – An `int` which specifies the number of variables considered.

Returns

An `int` array containing the identification numbers for the independent variables considered.

StepwiseRegression Class

Summary

Builds multiple linear regression models using forward selection, backward selection, or stepwise selection.

```
public class Imsl.Stat.StepwiseRegression
```

Properties

ANOVA

```
virtual public Imsl.Stat.ANOVA ANOVA {get; }
```

Description

An analysis of variance table and related statistics.

CoefficientTTests

```
virtual public Imsl.Stat.StepwiseRegression.CoefficientTTestsValue
  CoefficientTTests {get; }
```

Description

The student-t test statistics for the regression coefficients.

CoefficientVIF

```
virtual public double[] CoefficientVIF {get; }
```

Description

The variance inflation factors for the final model in this invocation.

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 *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

`virtual public double[,] CovariancesSwept {get; }`

Description

Results after *cov* has been swept for the columns corresponding to the variables in the model.

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

`virtual public int Force {set; }`

Description

Forces independent variables into the model based on their level assigned from `Levels`.

Variables with levels 1, 2, ..., `Force` are forced into the model as independent variables.

See Also: Levels (p. 370)

History

`virtual public double[] History {get; }`

Description

The stepwise regression history for the independent variables.

The last element corresponds to the dependent variable.

History[i]	Status of i-th Variable
0.0	This variable has never been added to the model.
0.5	This variable was added into the model during initialization.
$k > 0.0$	This variable was added to the model during the k-th step.
$k < 0.0$	This variable was deleted from model during the k-th step

See Also: Levels (p. 370)

Levels

`virtual public int[] Levels {set; }`

Description

The levels of priority for variables entering and leaving the regression.

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.

Default: 1, 1, ..., 1, -1 where -1 corresponds to the dependent variable.

See Also: Force (p. 369)

Method

`virtual public Imsl.Stat.StepwiseRegression.Direction Method {set; }`

Description

Specifies the stepwise selection method, forward, backward, or stepwise Regression.

Fields `Forward`, `Backward`, and `Stepwise` should be used.

Default: `Direction.Stepwise`.

See Also: Forward (p. 378), Backward (p. 378), Stepwise (p. 378)

PValueIn

`virtual public double PValueIn {set; }`

Description

Defines the largest p-value for variables entering the model.

Variables with p-value less than *PValueIn* may enter the model. Backward regression does not use this value.

Default: `PValueIn` = 0.05.

PValueOut

`virtual public double PValueOut {set; }`

Description

Defines the smallest p-value for removing variables.

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**PValueIn*. Forward regression does not use this value.

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.

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. 370)

Tolerance

`virtual public double Tolerance {set; }`

Description

The tolerance used to detect linear dependence among the independent variables.

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.

`Imsl.Stat.TooManyObsDeletedException` id is thrown if more observations have been deleted than were originally entered

`Imsl.Stat.MoreObsDelThanEnteredException` id is thrown if more observations are being deleted from the output covariance matrix than were originally entered

`Imsl.Stat.DiffObsDeletedException` id 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 *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.

`weights` – A `double` array containing the weight for each observation of x.

`Imsl.Stat.TooManyObsDeletedException` id is thrown if more observations have been deleted than were originally entered

`Imsl.Stat.MoreObsDelThanEnteredException` id is thrown if more observations are being deleted from the output covariance matrix than were originally entered

`Imsl.Stat.DiffObsDeletedException` id is thrown if different observations are being deleted from the return matrix than were originally entered

`Imsl.Stat.NegativeWeightException` id 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 *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.

`weights` – A `double` array containing the weight for each observation of x.

`frequencies` – A `double` array containing the frequency for each row of x.

`Imsl.Stat.TooManyObsDeletedException` id is thrown if more observations have been deleted than were originally entered

`Imsl.Stat.MoreObsDelThanEnteredException` id is thrown if more observations are being deleted from the output covariance matrix than were originally entered

`Imsl.Stat.DiffObsDeletedException` id is thrown if different observations are being deleted from the return matrix than were originally entered

`Imsl.Stat.NegativeWeightException` id is thrown if a weight is less than zero.

`Imsl.Stat.NegativeFreqException` id is thrown if a frequency is less than zero.

StepwiseRegression

`public StepwiseRegression(double[,] cov, int nObservations)`

Description

Creates a new instance of `StepwiseRegression` from a user-supplied variance-covariance matrix.

cov can be computed using the Imsl.Stat.Covariances (p. 257) class.

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*.

Method

Compute

`virtual public void Compute()`

Description

Builds the multiple linear regression models using forward selection, backward selection, or stepwise selection.

`Imsl.Stat.NoVariablesEnteredException` id is thrown if no variables entered the model. All elements of the Imsl.Stat.StepwiseRegression.ANOVA (p. 368) table are set to `NaN`

`Imsl.Stat.CyclingIsOccurringException` id is thrown if cycling occurs

Description

Class `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 Imsl.Stat.StepwiseRegression.Levels (p. 370). 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 Imsl.Stat.StepwiseRegression.Force (p. 369). 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 *cov*. Argument *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 *cov*. In this case, *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 *cov* are the same as in the previous case. Argument *nObservations* must be set to one greater than the number of observations.

The stepwise regression algorithm is due to Efroymson (1960). `StepwiseRegression` uses sweeps of the covariance matrix (input in *cov*, if the covariance matrix is specified, or generated internally) to move variables in and out of the model (Hemmerle 1967, Chapter 3). The 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 (`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.

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.

```
using System;
using Imsl.Math;
using Imsl.Stat;

public class StepwiseRegressionEx1
{
    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};

        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);
        pm.SetTitle("*** CovS *** "); pm.Print(pmf, sr.CovariancesSwept);
    }
}
```

Output

```
       *** ANOVA ***
               0
 0     2
 1    10
 2    12
 3  2641.00096476634
 4    74.7621121567348
 5  2715.76307692308
 6  1320.50048238317
 7     7.47621121567348
 8   176.62696308189
 9     1.58106023181431E-08
10    97.2471047716931
11    96.6965257260318
12     2.73426612012684
13     NaN
14     NaN

          *** Coef ***
    0       1       2       3
```

```
0   1.440  0.138   10.403  0.000
1   0.416  0.186   2.242   0.052
2  -0.410  0.199  -2.058   0.070
3  -0.614  0.049  -12.621  0.000

*** Swept ***
   0
0   1
1  -1
2  -1
3   1
4  -1

*** History ***
   0
0  2
1  0
2  0
3  1
4  0

     *** VIF ***
           0
0   1.0641052101769
1  18.780308640958
2   3.45960147891528
3   1.0641052101769

                 *** CovS ***
     0        1         2        3        4
0   0.003  -0.029    -0.946     0.000   1.440
1  -0.029   154.720  -142.800   0.907   64.381
2  -0.946  -142.800   142.302   0.070  -58.350
3   0.000   0.907     0.070     0.000  -0.614
4   1.440   64.381   -58.350   -0.614   74.762
```

StepwiseRegression.CoefficientTTestsValue Class

Summary

`CoefficientTTestsValue` contains statistics related to the student-t test, for each regression coefficient.

```
public class Imsl.Stat.StepwiseRegression.CoefficientTTestsValue
```

Methods

GetCoefficient

```
virtual public double GetCoefficient(int index)
```

Description

Returns the estimate for a coefficient of the independent variable.

index must be between 1 and the number of independent variables.

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.

GetPValue

```
virtual public double GetPValue(int index)
```

Description

Returns the p-value for the two-sided test $H_0 : \beta = 0$ vs. $H_1 : \beta \neq 0$.

index must be between 1 and the number of independent variables.

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.

GetStandardError

```
virtual public double GetStandardError(int index)
```

Description

Returns the estimated standard error for a coefficient estimate.

index must be between 1 and the number of independent variables.

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.

GetTStatistic

```
virtual public double GetTStatistic(int index)
```

Description

Returns the student-t test statistic for testing the i-th coefficient equal to zero ($\beta_{index} = 0$).

index must be between 1 and the number of independent variables.

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.

StepwiseRegression.Direction Enumeration

Summary

Direction indicator.

`public enumeration Imsl.Stat.StepwiseRegression.Direction`

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 intitialization, 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

Summary

Generates summary statistics using user-supplied functions in a nonlinear regression model.

`public class Imsl.Stat.UserBasisRegression`

Property

ANOVA

`public Imsl.Stat.ANOVA ANOVA {get; }`

Description

An analysis of variance table and related statistics.

Constructor

UserBasisRegression

```
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 `boolean` which specifies whether or not the model has an intercept.

Methods

GetCoefficients

```
public double[] GetCoefficients()
```

Description

Returns the regression coefficients.

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.

Returns

A `double` array containing the regression coefficients.

`Imsl.Math.SingularMatrixException` id is thrown when the regression matrix is singular

Update

```
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.

```
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];
        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();
        new PrintMatrix
            ("The regression coefficients are:").Print(coef);
    }
}
```

Output

```
The regression coefficients are:
           0
0  1.01010532376649
1  0.0199013147736359
2  7.02909074858517
3  0.0374000977854433
```

IRegressionBasis Interface

Summary

Interface for user supplied function to `UserBasisRegression` object.

`public interface Imsl.Stat.IRegressionBasis`

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

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.

ANOVA Class

Summary

Analysis of Variance table and related statistics.

`public class Imsl.Stat.ANOVA`

Properties

AdjustedRSquared

`public double AdjustedRSquared {get; }`

Description

Returns the adjusted R-squared (in percent).

CoefficientOfVariation

`public double CoefficientOfVariation {get; }`

Description

Returns the coefficient of variation (in percent).

DegreesOfFreedomForError

`public double DegreesOfFreedomForError {get; }`

Description

Returns the degrees of freedom for error.

DegreesOfFreedomForModel

`public double DegreesOfFreedomForModel {get; }`

Description

Returns the degrees of freedom for the model.

ErrorMeanSquare

`public double ErrorMeanSquare {get; }`

Description

Returns the error mean square.

F

`public double F {get; }`

Description

Returns the F statistic.

MeanOfY

`public double MeanOfY {get; }`

Description

Returns the mean of the response (dependent variable).

ModelErrorStdev

`public double ModelErrorStdev {get; }`

Description

Returns the estimated standard deviation of the model error.

ModelMeanSquare

`public double ModelMeanSquare {get; }`

Description

Returns the model mean square.

P

`public double P {get; }`

Description

Returns the p-value.

RSquared

`public double RSquared {get; }`

Description

Returns the R-squared (in percent).

SumOfSquaresForError

`public double SumOfSquaresForError {get; }`

Description

Returns the sum of squares for error.

SumOfSquaresForModel

`public double SumOfSquaresForModel {get; }`

Description

Returns the sum of squares for model.

TotalDegreesOfFreedom

`public double TotalDegreesOfFreedom {get; }`

Description

Returns the total degrees of freedom.

TotalMissing

`public int TotalMissing {get; }`

Description

Returns the total number of missing values.

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.

Constructors

ANOVA

`public ANOVA(double[][] y)`

Description

Analyzes a one-way classification model.

The rows in y correspond to observation groups. Each row of y can contain a different number of observations.

Parameter

`y` – Two-dimension `double` array containing the responses.

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.

If the grand mean is not known it may be set to not-a-number.

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.

Methods

GetArray

`public double[] GetArray()`

Description

Returns the ANOVA values as an array.

Returns

A `double`[15] array containing the following values.

index	Value
0	Degrees of freedom for model
1	Degrees of freedom for error
2	Total degrees of freedom
3	Sum of squares for model
4	Sum of squares for error
5	Total sum of squares
6	Model mean square
7	Error mean square
8	F statistic
9	p-value
10	R-squared (in percent)
11	Adjusted R-squared (in percent)
12	Estimated standard deviation of the model error
13	Mean of the response (dependent variable)
14	Coefficient of variation (in percent)

GetDunnSidak

`public double GetDunnSidak(int i, int j)`

Description

Computes the confidence intervals on i-th mean - j-th mean using the Dunn-Sidak method.

Parameters

`i` – An `int` indicating the i-th mean, μ_i.

`j` – An `int` containing the j-th mean μ_j.

Returns

The confidence intervals on i-th mean - j-th mean using the Dunn-Sidak method.

GetGroupInformation

`public double[][] GetGroupInformation()`

Description

Returns information concerning the groups.

Row i contains information pertaining to the i-th group. The information in the columns is as follows:

Column	Information
0	Group Number
1	Number of nonmissing observations
2	Group Mean
3	Group Standard Deviation

Returns

A two-dimensional `double` array containing information concerning the groups.

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).

```
using System;
using Imsl.Stat;
using Imsl.Math;


public class ANOVAEx1
{
    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.Out.WriteLine
            ("Degrees Of Freedom For Model = " + aov[0]);
        Console.Out.WriteLine
            ("Degrees Of Freedom For Error = " + aov[1]);
        Console.Out.WriteLine
            ("Total (Corrected) Degrees Of Freedom = " + aov[2]);
        Console.Out.WriteLine("Sum Of Squares For Model = " + aov[3]);
        Console.Out.WriteLine("Sum Of Squares For Error = " + aov[4]);
        Console.Out.WriteLine
            ("Total (Corrected) Sum Of Squares = " + aov[5]);
        Console.Out.WriteLine("Model Mean Square = " + aov[6]);
```

```
        Console.Out.WriteLine("Error Mean Square = " + aov[7]);
        Console.Out.WriteLine("F statistic = " + aov[8]);
        Console.Out.WriteLine("P value= " + aov[9]);
        Console.Out.WriteLine("R Squared (in percent) = " + aov[10]);
        Console.Out.WriteLine
            ("Adjusted R Squared (in percent) = " + aov[11]);
        Console.Out.WriteLine
            ("Model Error Standard deviation = " + aov[12]);
        Console.Out.WriteLine("Mean Of Y = " + aov[13]);
        Console.Out.WriteLine
            ("Coefficient Of Variation (in percent) = " + aov[14]);
        Console.Out.WriteLine
            ("Total number of missing values = " + anova.TotalMissing);

        PrintMatrixFormat pmf = new PrintMatrixFormat();
        String[] labels =
            new String[]{"Group", "N", "Mean", "Std. Deviation"};
        pmf.SetColumnLabels(labels);
        pmf.NumberFormat = null;
        new PrintMatrix("Group Information").Print(pmf,
            (Object)anova.GetGroupInformation());
    }
}
```

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
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
   Group  N  Mean   Std. Deviation
0    0    3  100   5.56776436283002
1    1    2   86   2.82842712474619
2    2    1   32   NaN
```

ANOVAFactorial Class

Summary

Analyzes a balanced factorial design with fixed effects.

`public class Imsl.Stat.ANOVAFactorial`

Properties

ErrorIncludeType

`public Imsl.Stat.ANOVAFactorial.ErrorCalculation ErrorIncludeType {get; set; }`

Description

The error included type.

`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.

`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.

Default: `ModelOrder` = `nSubscripts`-1

Constructor

ANOVAFactorial

`public ANOVAFactorial(int nSubscripts, int[] nLevels, double[] y)`

Description

Constructor for `ANOVAFactorial`.

`y` must not contain `NaN` for any of its elements; i.e., missing values are not allowed.

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.

System.ArgumentException id is thrown if nLevels.length, and y.length are not consistent

Methods

Compute

`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

`public double[] GetANOVATable()`

Description

Returns the analysis of variance table.

The analysis of variance statistics are given as follows:

Element	Analysis of Variance Statistics
0	Degrees of freedom for the model
1	Degrees of freedom for error
2	Total (corrected) degrees of freedom
3	Sum of squares for the model
4	Sum of squares for error
5	Total (corrected) sum of squares
6	Model mean square
7	Error mean square
8	Overall F-statistic
9	p-value
10	R^2 (in percent)
11	Adjusted R^2 (in percent)
12	Estimate of the standard deviation
13	Overall mean of y
14	Coefficient of variation (in percent)

Returns

A `double` array containing the analysis of variance table.

GetMeans

`public double[] GetMeans()`

Description

Returns the subgroup means.

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.

Here,

$$\mathrm{NEF} = \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{\min(n, |\mathrm{model\_order}|)}$$

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, AB, AC, BC, and ABC, respectively.

The columns of the output matrix are as follows:

Column	Description
0	Degrees of freedom
1	Sum of squares
2	F-statistic
3	p-value

Returns

A `double` matrix containing statistics relating to the sums of squares for the effects in the model.

Description

Class `ANOVAFactorial` performs an analysis for an n-way classification design with balanced data. For balanced data, here 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.

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

$$y_{ijk} = \mu + \alpha_i + \beta_j + \gamma_{ij} + \varepsilon_{ijk} \quad i = 1,\ 2;\ j = 1,\ 2,\ 3;\ k = 1,\ 2,\ \ldots,\ 10$$

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:

	Protein Source (A)		
Protein Level (B)	**Beef**	**Cereal**	**Pork**
High	73, 102, 118, 104, 81, 107, 100, 87, 117, 111	98, 74, 56, 111, 95, 88, 82, 77, 86, 92	94, 79, 96, 98, 102, 102, 108, 91, 120, 105
Low	90, 76, 90, 64, 86, 51, 72, 90, 95, 78	107, 95, 97, 80, 98, 74, 74, 67, 89, 58	49, 82, 73, 86, 81, 97, 106, 70, 61, 82

```
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, 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};

        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                              ",
            "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 = new String[]{"A", "B", "A*B"};
        String[] mlabels = new String[]{"grand mean     ",
            "A1             ", "A2             ", "A3             ",
            "B1             ", "B2             ", "A1*B1         ",
            "A1*B2          ", "A2*B1          ", "A2*B2          ",
            "A3*B1          ", "A3*B2          "};

        ANOVAFactorial af =
            new ANOVAFactorial(nSubscripts, nLevels, y);

        Console.Out.WriteLine
            ("P-value = " + af.Compute().ToString("0.000000"));

        Console.Out.WriteLine
            ("\n          * * * Analysis of Variance * * *");
        double[] anova = af.GetANOVATable();
        for (i = 0; i < anova.Length; i++)
        {
            Console.Out.WriteLine
                (labels[i] + " " + anova[i].ToString("0.0000"));
        }

        Console.Out.WriteLine
            ("\n          * * * Variation Due to the " + "Model * * *");
        Console.Out.WriteLine
            ("Source\tDF\tSum of Squares\tMean Square" +
             "\tProb. of Larger F");
        double[,] te = af.GetTestEffects();
        for (i = 0; i < te.GetLength(0); i++)
        {
            Console.Out.WriteLine(
                rlabels[i] + "\t" +
                te[i,0].ToString("0.0000") + "\t" +
                te[i,1].ToString("0.0000") + "\t" +
                te[i,2].ToString("0.0000") + "\t\t" +
                te[i,3].ToString("0.0000"));
        }

        Console.Out.WriteLine("\n* * * Subgroup Means * * *");
        double[] means = af.GetMeans();
        for (i = 0; i < means.Length; i++)
        {
            Console.Out.WriteLine
                (mlabels[i] + " " + means[i].ToString("0.0000"));
        }
    }
}
```

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 * * *
Source DF Sum of Squares Mean Square Prob. of Larger F
A 2.0000 266.5333 0.6211 0.5411
B 1.0000 3168.2667 14.7666 0.0003
A*B 2.0000 1178.1333 2.7455 0.0732

* * * 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
A1*B2           79.2000
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:

	A_0		
	B_0	B_1	B_2
C_0	88.76	91.41	97.85
C_1	87.45	98.27	95.85
C_2	86.01	104.20	90.09

	A_1		
	B_0	B_1	B_2
C_0	94.83	100.49	99.75
C_1	84.57	97.20	112.30
C_2	81.06	120.80	108.77

	A_2		
	B_0	B_1	B_2
C_0	99.90	100.23	104.50
C_1	92.98	107.77	110.94
C_2	94.72	118.39	102.87

```
using System;
using Imsl.Stat;

public 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, 95.85, 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 =
            new String[]{"A", "B", "C", "A*B", "A*C", "B*C"};

        ANOVAFactorial af =
            new ANOVAFactorial(nSubscripts, nLevels, y);
```

```
        af.ErrorIncludeType = ANOVAFactorial.ErrorCalculation.Pooled;
        Console.Out.WriteLine
            ("P-value = " + af.Compute().ToString("0.000000"));

        Console.Out.WriteLine
            ("\n          * * * Analysis of Variance * * *");
        double[] anova = af.GetANOVATable();
        for (i = 0; i < anova.Length; i++)
        {
            Console.Out.WriteLine
                (labels[i] + " " + anova[i].ToString("0.0000"));
        }

        Console.Out.WriteLine
            ("\n          * * * Variation Due to the " + "Model * * *");
        Console.Out.WriteLine
            ("Source\tDF\tSum of Squares\tMean Square" +
             "\tProb. of Larger F");
        double[,] te = af.GetTestEffects();
        for (i = 0; i < te.GetLength(0); i++)
        {
            System.Text.StringBuilder sb =
                new System.Text.StringBuilder(rlabels[i]);

            int len = sb.Length;
            for (int j = 0; j < (8 - len); j++)
                sb.Append(' ');
            sb.Append(te[i,0].ToString("0.0000"));

            len = sb.Length;
            for (int j = 0; j < (16 - len); j++)
                sb.Append(' ');
            sb.Append(te[i,1].ToString("0.0000"));

            len = sb.Length;
            for (int j = 0; j < (32 - len); j++)
                sb.Append(' ');
            sb.Append(te[i,2].ToString("0.0000"));

            len = sb.Length;
            for (int j = 0; j < (48 - len); j++)
                sb.Append(' ');
            sb.Append(te[i,3].ToString("0.0000"));

            Console.Out.WriteLine(sb.ToString());
        }
    }
}
```

Output

```
P-value = 0.008299
```

```
          * * * Analysis of Variance * * *
degrees of freedom for the model                   18.0000
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 * * *
Source DF Sum of Squares Mean Square Prob. of Larger F
A       2.0000  488.3675        10.5152        0.0058
B       2.0000  1090.6564       23.4832        0.0004
C       2.0000  49.1485         1.0582         0.3911
A*B     4.0000  142.5853        1.5350         0.2804
A*C     4.0000  32.3474         0.3482         0.8383
B*C     4.0000  592.6238        6.3800         0.0131
```

ANOVAFactorial.ErrorCalculation Enumeration

Summary

ErrorCalculation members indicate whether interaction effects are pooled into the error or not.

`public enumeration Imsl.Stat.ANOVAFactorial.ErrorCalculation`

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

Summary

Performs Student-Newman-Keuls multiple comparisons test.

```
public class Imsl.Stat.MultipleComparisons
```

Property

Alpha

```
public double Alpha {get; set; }
```

Description

The significance level of the test

`Alpha` must be in the interval [0.01, 0.10]. Default: `Alpha` = 0.01

Constructor

MultipleComparisons

```
public MultipleComparisons(double[] means, int df, double stdError)
```

Description

Constructor for `MultipleComparisons`.

In fixed effects models, `stdError` equals the estimated standard error of a mean. For example, in a one-way model stdError = $\sqrt{s^2/n}$ where s^2 is the estimate of σ^2 and n is the number of responses in a sample mean. In models with random components, use stdError = sedif/$\sqrt{2}$ where `sedif` is the estimated standard error of the difference of two means.

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.

Method

Compute

```
public int[] Compute()
```

Description

Performs Student-Newman-Keuls multiple comparisons test.

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.

Returns

A `int` array , call it `equalMeans`, indicating the size of the groups of means declared to be equal.

Description

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).

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).

```
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

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

Summary

Performs a chi-squared analysis of a two-way contingency table.

```
public class Imsl.Stat.ContingencyTable
```

Properties

ChiSquared

```
public double ChiSquared {get; }
```

Description

Returns the Pearson chi-squared test statistic.

ContingencyCoef

`public double ContingencyCoef {get; }`

Description

Returns contingency coefficient.

CramersV

`public double CramersV {get; }`

Description

Returns Cramer's V.

DegreesOfFreedom

`public int DegreesOfFreedom {get; }`

Description

Returns the degrees of freedom for the chi-squared tests associated with the table.

ExactMean

`public double ExactMean {get; }`

Description

Returns the exact mean.

ExactStdev

`public double ExactStdev {get; }`

Description

Returns the exact standard deviation.

GSquared

`public double GSquared {get; }`

Description

Returns the likelihood ratio G^2 (chi-squared).

GSquaredP

`public double GSquaredP {get; }`

Description

Returns the probability of a larger G^2 (chi-squared).

P

```
public double P {get; }
```

Description

Returns the Pearson chi-squared p-value for independence of rows and columns.

Phi

```
public double Phi {get; }
```

Description

Returns phi.

Constructor

ContingencyTable

```
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

```
public double[,] GetContributions()
```

Description

Returns the contributions to chi-squared for each cell in the table.

The last row and column contain the total contribution to chi-squared for that row or column.

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.

GetExpectedValues

```
public double[,] GetExpectedValues()
```

Description

Returns the expected values of each cell in the table.

The marginal totals are in the last row and column.

Returns

A `double` matrix of size (`table.GetLength(0)`+1) * (`table.GetLength(1)`+1) containing the expected values of each cell in the table, under the null hypothesis.

GetStatistics

`public double[,] GetStatistics()`

Description

Returns the statistics associated with this table.

Each row corresponds to a statistic.

Row	Statistics
0	gamma
1	Kendall's τ_b
2	Stuart's τ_c
3	Somers' D for rows (given columns)
4	Somers' D for columns (given rows)
5	product moment correlation
6	Spearman rank correlation
7	Goodman and Kruskal τ for rows (given columns)
8	Goodman and Kruskal τ for columns (given rows)
9	uncertainty coefficient U (symmetric)
10	uncertainty $U_{r\|c}$ (rows)
11	uncertainty $U_{c\|r}$ (columns)
12	optimal prediction λ (symmetric)
13	optimal prediction $\lambda_{r\|c}$ (rows)
14	optimal prediction $\lambda_{c\|r}$ (columns)
15	optimal prediction $\lambda^*_{r\|c}$ (rows)
16	optimal prediction $\lambda^*_{c\|r}$ (columns)
17	Test for linear trend in row probabilities if `table.GetLength(0)` = 2. Test for linear trend in column probabilities if `table.GetLength(1)` = 2 and `table.GetLength(0)` is not 2
18	Kruskal-Wallis test for no row effect
19	Kruskal-Wallis test for no column effect
20	kappa (square tables only)
21	McNemar test of symmetry (square tables only)
22	McNemar one degree of freedom test of symmetry (square tables only)

The columns are as follows:

Column	Value
0	estimated statistic
1	standard error for any parameter value
2	standard error under the null hypothesis
3	t value for testing the null hypothesis
4	p-value of the test in column 3

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.

Returns

A `double` matrix of size 23 * 5 containing statistics associated with this table.

Description

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 $p_{ij} = p_{i\bullet}p_{j\bullet}$ denote the predicted cell probabilities under the null hypothesis of independence, where $p_{i\bullet}$ and $p_{j\bullet}$ 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 χ^2 is given as $(x_{ij} - e_{ij})^2/e_{ij}$. The Pearson chi-squared statistic (denoted χ^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., $H_0 : p_{ij} = p_{i\bullet}p_{j\bullet}$. The null hypothesis is rejected if the computed value of χ^2 is too large.

The maximum likelihood equivalent of χ^2, G^2 is computed as follows:

$$G^2 = -2\sum_{i,j} x_{ij} \ln\left(x_{ij}/np_{ij}\right)$$

G^2 is asymptotically equivalent to χ^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:

$$\text{phi}, \ \phi = \sqrt{\chi^2/n}$$

$$\text{contingency coefficient}, \ P = \sqrt{\chi^2/\left(n+\chi^2\right)}$$

$$\text{Cramer's}\, V, \ V = \sqrt{\chi^2/\left(n \min\left(r,c\right)\right)}$$

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 χ^2 statistic, which is contained in the `ChiSquared` property.

The distribution of the χ^2 statistic in finite samples approximates a chi-squared distribution. To compute the exact mean and standard deviation of the χ^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 `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, ϕ, P, and V, do not require any ordering of the row and column categories. Class `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 τ_b, Stuart's τ_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 τ_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 τ_b cannot be 1.0 (if all marginal totals are positive). For this reason, Stuart suggested a modification to the denominator of τ 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 τ_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

$$\lambda_{c\,|\,r} = \frac{(1 - p_{\bullet m}) - (1 - \sum_i p_{im})}{1 - p_{\bullet m}}$$

where m is the index of the maximum estimated probability in the row (p_{im}) or row margin ($p_{\bullet m}$). A similar coefficient is defined for predicting the rows from the columns. The symmetric version of the optimal prediction λ 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 λ 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 λ coefficients in which first the row (or column) marginals are adjusted to the same number of observations. This yields

$$\lambda^*_{c\,|\,r} = \frac{\sum_i \max_j p_{j\,|\,i} - \max_j(\sum_i p_{j\,|\,i})}{R - \max_j(\sum_i p_{j\,|\,i})}$$

where i indexes the rows, j indexes the columns, and $p_{j|i}$ is the (estimated) probability of column j given row i.

$$\lambda^*_{r\,|\,c}$$

is similarly defined.

Goodman and Kruskal τ: 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:

$$n/2 - (\sum_i x^2_{i\bullet})/\,(2n)$$

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 τ 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:

$$q_j = x_{\bullet j}/2 - (\sum_i x_{ij}^2)/(2x_{i\bullet})$$

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 τ 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:

$$U_{r|c} = \frac{\sum_{i,j} x_{ij} \log(x_{i\bullet} x_{\bullet j}/nx_{ij})}{\sum_i x_{i\bullet} \log(x_{i\bullet}/n)}$$

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

$$\hat{\beta} = \frac{\sum_j x_{\bullet j} (x_{1j}/x_{\bullet j} - x_{1\bullet}/n)(j - \bar{j})}{\sum_j x_{\bullet j} (j - \bar{j})^2}$$

where

$$\bar{j} = \sum_j x_{\bullet j} j/n$$

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

$$p_0 = \sum_i x_{ii}/n$$

denote the probability that the two judges agree, and let

$$p_c = \sum_i e_{ii}/n$$

denote the expected probability of agreement under the independence model. Kappa is then given by $(p_0 - p_c)/(1 - p_c)$.

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.

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.

```
using System;
using Imsl.Stat;
```

```
public 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.

```
using System;
using Imsl.Stat;
using Imsl.Math;

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("* * * Expected Values * * *");
pm.Print(pmf, ct.GetExpectedValues());

pmf.NumberFormat = "0.0000";
pm.SetTitle("* * * Contributions to Chi-squared* * *");
pm.Print(pmf, ct.GetContributions());

Console.Out.WriteLine("* * * 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 * * *
     0       1       2       3
0  821.00  112.00  85.00   35.00
1  116.00  494.00  145.00  27.00
2  72.00   151.00  583.00  87.00
3  43.00   34.00   106.00  331.00

         * * * Expected Values * * *
      0        1       2       3        4
0  341.69   256.92  298.49  155.90  1053.00
1  253.75   190.80  221.67  115.78  782.00
2  289.77   217.88  253.14  132.21  893.00
3  166.79   125.41  145.70  76.10   514.00
4  1052.00  791.00  919.00  480.00  3242.00

        * * * Contributions to Chi-squared* * *
       0         1         2          3          4
0  672.3626   81.7416   152.6959  93.7612    1000.5613
1  74.7802    481.8351  26.5189   68.0768    651.2109
2  163.6605   20.5287   429.8489  15.4625    629.5006
3  91.8743    66.6263   10.8183   853.7768   1023.0957
4  1002.6776  650.7317  619.8819  1031.0772  3304.3684

* * * Chi-square Statistics * * *
```

```
Exact mean = 9.0028
Exact standard deviation = 4.2402
Phi = 1.0096
P = 0.7105
Cramer's V = 0.5829

              stat.      std. err.   std. err.(Ho) t-value(Ho)  p-value
Gamma         0.7757     0.0123      0.0149        52.1897      0.0000
Tau B         0.6429     0.0122      0.0123        52.1897      0.0000
Tau C         0.6293     0.0121      NaN           52.1897      0.0000
D-Row         0.6418     0.0122      0.0123        52.1897      0.0000
D-Column      0.6439     0.0122      0.0123        52.1897      0.0000
Correlation   0.6926     0.0128      0.0172        40.2669      0.0000
Spearman      0.6939     0.0127      0.0127        54.6614      0.0000
GK tau rows   0.3420     0.0123      NaN           NaN          NaN
GK tau cols.  0.3430     0.0122      NaN           NaN          NaN
U - sym.      0.3171     0.0110      NaN           NaN          NaN
U - rows      0.3178     0.0110      NaN           NaN          NaN
U - cols.     0.3164     0.0110      NaN           NaN          NaN
Lambda-sym.   0.5373     0.0124      NaN           NaN          NaN
Lambda-row    0.5374     0.0126      NaN           NaN          NaN
Lambda-col.   0.5372     0.0126      NaN           NaN          NaN
l-star-rows   0.5506     0.0136      NaN           NaN          NaN
l-star-col.   0.5636     0.0127      NaN           NaN          NaN
Lin. trend    NaN        NaN         NaN           NaN          NaN
Kruskal row   1561.4859  3.0000      NaN           NaN          0.0000
Kruskal col.  1563.0303  3.0000      NaN           NaN          0.0000
Kappa         0.5744     0.0111      0.0106        54.3583      0.0000
McNemar       4.7625     6.0000      NaN           NaN          0.5746
McNemar df=1  0.9487     1.0000      NaN           0.3459       0.3301
```

CategoricalGenLinModel Class

Summary

Analyzes categorical data using logistic, probit, Poisson, and other linear models.

`public class Imsl.Stat.CategoricalGenLinModel`

Properties

CaseAnalysis

`virtual public double[,] CaseAnalysis {get; }`

Description

The case analysis.

The matrix is $nobs \times 5$ where $nobs$ is the number of observations. The matrix contains:

Column	Statistic
0	Prediction.
1	The residual.
2	The estimated standard error of the residual.
3	The estimated influence of the observation.
4	The standardized residual.

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:

Model	Prediction
0	The predicted mean for the observation.
1-4	The probability of a success on a single trial.

CensorColumn

`virtual public int CensorColumn {set; }`

Description

The column number in x which contains the interval type for each observation.

The valid codes are interpreted as:

`x[i,CensorColumn]`	Censoring
0	Point observation. The response is unique and is given by `x[i,LowerEndpointColumn]`.
1	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.
2	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.
3	Full interval. The response is greater than or equal to `x[i,LowerEndpointColumn]` but less than or equal to `x[i,UpperEndpointColumn]`.

Default: `CensorColumn` = 0.

ClassificationVariableColumn

`virtual public int[] ClassificationVariableColumn {set; }`

Description

An index vector to contain the column numbers in x that are classification variables.

By default this vector is not referenced.

ClassificationVariableCounts

`virtual public int[] ClassificationVariableCounts {get; }`

Description

The number of values taken by each classification variable.

ClassificationVariableValues

`virtual public double[] ClassificationVariableValues {get; }`

Description

The distinct values of the classification variables in ascending order.

A `null` is returned if Imsl.Stat.CategoricalGenLinModel.Solve (p. 427) has not been called prior to calling this method.

ConvergenceTolerance

`virtual public double ConvergenceTolerance {set; }`

Description

The convergence criterion.

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. 423), from one iteration to the next is less than `ConvergenceTolerance`/100. `ConvergenceTolerance` must be greater than 0.

Default: `ConvergenceTolerance` = .001.

CovarianceMatrix

`virtual public double[,] CovarianceMatrix {get; }`

Description

The estimated asymptotic covariance matrix of the coefficients.

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.

ExtendedLikelihoodObservations

`virtual public int[] ExtendedLikelihoodObservations {get; set; }`

Description

A vector indicating which observations are included in the extended likelihood.

`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:

Value	Status of observation
0	Observation i is in the likelihood.
1	Observation i cannot be in the likelihood because it contains at least one missing value in x.
2	Observation i is not in the likelihood. Its estimated parameter is infinite.

A `null` is returned if Imsl.Stat.CategoricalGenLinModel.Solve (p. 427) has not been called prior to calling this method.

Default: All elements are zero.

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.

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.

FrequencyColumn

```
virtual public int FrequencyColumn {set; }
```

Description

The column number in x that contains the frequency of response for each observation.

By default a frequency of 1 for each observation is assumed.

Hessian

```
virtual public double[,] Hessian {get; }
```

Description

The Hessian computed at the initial parameter estimates.

The Hessian matrix is *nCoef* by *nCoef* where *nCoef* is the number of coefficients in the model. This member function will call Imsl.Stat.CategoricalGenLinModel.Solve (p. 427) to get the Hessian if the Hessian has not already been computed.

InfiniteEstimateMethod

```
virtual public int InfiniteEstimateMethod {set; }
```

Description

Specifies the method used for handling infinite estimates.

The value of `InfiniteEstimateMethod` is interpreted as follows:

`InfiniteEstimateMethod`	**Method**
0	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.
1	Iterate without checking for infinite estimates.

By default `InfiniteEstimateMethod` = 1.

LastParameterUpdates

```
virtual public double[] LastParameterUpdates {get; }
```

Description

The last parameter updates (excluding step halvings).

LowerEndpointColumn

```
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.

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.

MaxIterations

```
virtual public int MaxIterations {set; }
```

Description

The maximum number of iterations allowed.

Default: `MaxIterations` = 30.

ModelIntercept

`virtual public int ModelIntercept {set; }`

Description

The intercept option.

Input `ModelIntercept` is interpreted as follows:

Value	Action
0	No intercept is in the model (unless otherwise provided for by the user).
1	Intercept is automatically included in the model.

By default `ModelIntercept` = 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.

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

`virtual public int ObservationMax {set; }`

Description

The maximum number of observations that can be handled in the linear programming.

Default: `ObservationMax` is set to the number of observations.

OptimizedCriterion

`virtual public double OptimizedCriterion {get; }`

Description

The optimized criterion.

The criterion to be maximized is a constant plus the log-likelihood.

OptionalDistributionParameterColumn

`virtual public int OptionalDistributionParameterColumn {set; }`

Description

The column number in x that contains an optional distribution parameter for each observation.

The distribution parameter values are interpreted as follows depending on the model chosen:

Model	Meaning of `x[i,OptionalDistributionParameterColumn]`
0	The Poisson parameter is given by $x[i, OptionalDistributionParameterColumn] \times e^{\rho}$.
1	The number of successes required in the negative binomial is given by `x[i,OptionalDistributionParameterColumn].`
2	`x[i,OptionalDistributionParameterColumn]` is not used.
3-5	The number of trials in the binomial distribution is given by `x[i,OptionalDistributionParameterColumn].`

By default the distribution parameter is assumed to be 1.

Parameters

`virtual public double[,] Parameters {get; }`

Description

Parameter estimates and associated statistics.

Here, *nCoef* is the number of coefficients in the model. The statistics returned are as follows:

Column	**Statistic**
0	Coefficient estimate.
1	Estimated standard deviation of the estimated coefficient.
2	Asymptotic normal score for testing that the coefficient is zero.
3	ρ - value associated with the normal score in column 2.

Product

`virtual public double[] Product {get; }`

Description

The inverse of the Hessian times the gradient vector computed at the input parameter estimates.

nCoef is the number of coefficients in the model. This member function will call Imsl.Stat.CategoricalGenLinModel.Solve (p. 427) to get the product if the product has not already been computed.

UpperBound

`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.

Default: `UpperBound` = 1.

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.

By default all observations are treated as "point" observations.

Constructor

CategoricalGenLinModel

```
public CategoricalGenLinModel(double[,] x,
  Imsl.Stat.CategoricalGenLinModel.DistributionParameterModel model)
```

Description

Constructs a new `CategoricalGenLinModel`.

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:

Model	Distribution	Function	Lower-bound
0	Poisson	Exponential	0
1	Negative Binomial	Logistic	0
2	Logarithmic	Logistic	1
3	Binomial	Logistic	0
4	Binomial	Probit	0
5	Binomial	Log-log	0

Let γ 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:

Name	Function
Exponential	e^{γ}
Logistic	$e^{\gamma}/(1+e^{\gamma})$
Probit	$\Phi(\gamma)$ (where Φ is the normal cdf)
Log-log	$1-e^{-\gamma}$

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.

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.

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 `nvef[0]` elements of *indef* give the column numbers of the variables in the first effect. The next `nvef[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.

Parameters

`indef` – An `int` vector of length $\sum_{k=0}^{\mathrm{nef}-1} \mathrm{nvef[k]}$ where *nef* is the number of effects in the model.

`nvef` – An `int` vector of length *nef* where *nef* is the number of effects in the model.

`System.ArgumentException` id 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.

If this method is not called, `init` is set to 0.

`init`	**Action**
0	Unweighted linear regression is used to obtain initial estimates.
1	The *nCoef*, number of coefficients, elements of *estimates* contain initial estimates of the parameters. Use of this option requires that the user know *nCoef* beforehand.

estimates is used if `init = 1`. If this member function is not called, unweighted linear regression is used to obtain the initial estimates.

Parameters

`init` – An input `int` indicating the desired initialization method for the initial estimates of the parameters.

`estimates` – An input `double` array of length *nCoef* containing the initial estimates of the parameters where *nCoef* is the number of estimated coefficients in the model.

`System.ArgumentException` id is thrown when `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.

Here, *nCoef* is the number of coefficients in the model. The statistics returned are as follows:

Column	Statistic
0	Coefficient estimate.
1	Estimated standard deviation of the estimated coefficient.
2	Asymptotic normal score for testing that the coefficient is zero.
3	ρ - value associated with the normal score in column 2.

Returns

An *nCoef* row by 4 column `double` matrix containing the parameter estimates and associated statistics.

`Imsl.Stat.ClassificationVariableException` id 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` id 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. 424)

`Imsl.Stat.DeleteObservationsException` id is thrown if the number of observations to delete has grown too large

Description

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, β 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 γ_i 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 $\hat{\gamma}_i$. These estimates, when combined with the set of indices of the observations such that $\hat{\gamma}_i$ is infinite at the supremum of the likelihood, are called extended maximum estimates. When none of the optimal $\hat{\gamma}_i$ 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. 421) = 0. See below.

The models available in CategoricalGenLinModel are:

Model Name	**Parameterization**	**Response PDF**
Model0 (Poisson)	$\lambda = N \times e^{w+\eta}$	$f(y) = \lambda^y e^{-\lambda}/y!$
Model1 (Negative Binomial)	$\theta = \frac{e^{w+\eta}}{1+e^{w+\eta}}$	$f(y) = \begin{pmatrix} S+y-1 \\ y-1 \end{pmatrix} \theta^S (1-\theta)^y$
Model2 (Logarithmic)	$\theta = \frac{e^{w+\eta}}{1+e^{w+\eta}}$	$f(y) = (1-\theta)^y/(y \ln \theta)$
Model3 (Logistic)	$\theta = \frac{e^{w+\eta}}{1+e^{w+\eta}}$	$f(y) = \begin{pmatrix} N \\ y \end{pmatrix} \theta^y (1-\theta)^{N-y}$
Model4 (Probit)	$\theta = \Phi(w+\eta)$	$f(y) = \begin{pmatrix} N \\ y \end{pmatrix} \theta^y (1-\theta)^{N-y}$
Model5 (Log-log)	$\theta = 1 - e^{-e^{w+\eta}}$	$f(y) = \begin{pmatrix} N \\ y \end{pmatrix} \theta^y (1-\theta)^{N-y}$

Here Φ denotes the cumulative normal distribution, N and S are known parameters specified for each observation via column Imsl.Stat.CategoricalGenLinModel.OptionalDistributionParameterColumn (p. 423) of x, and w is an optional fixed parameter specified for each observation via column Imsl.Stat.CategoricalGenLinModel.FixedParameterColumn (p. 421) 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 θ, θ 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. 421) of the matrix x, each vector can represent several observations. Also note that classification variables and their products are easily incorporated into the models via the usual regression-type specifications.

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,OptionalDistributionParameterColumn]` and `x[i,FrequencyColumn]`. In all cases these values default to 1. Means are computed as

$$\bar{x} = \frac{\Sigma_i f_i x_i}{\Sigma_i f_i}$$

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, θ for point observations may be estimated as

$$\hat{\theta} = x[i, LowerEndpointColumn]/x[i, OptionalDistributionParameterColumn]$$

   and, when $model = 3$, the linear relationship is given by

$$\left(\ln(\hat{\theta}/(1-\hat{\theta})) \approx x\beta\right)$$

   while if `model` $= 4$,

$$\left(\Phi^{-1}(\hat{\theta}) = x\beta\right)$$

   For bounded interval observations, the midpoint of the interval is used for `x[i,Imsl.Stat.CategoricalGenLinModel.LowerEndpointColumn (p. 422)]`. 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

$$\Psi(x_i^T\beta)$$

denote the log of the probability of the i-th observation for coefficients β. In the least-squares model, the weight of the i-th observation is taken as the absolute value of the second derivative of

$$\Psi(x_i^T\beta)$$

with respect to

$$\gamma_i = x_i^T\beta$$

(times the frequency of the observation), and the dependent variable is taken as the first derivative Ψ with respect to γ_i, divided by the square root of the weight times the frequency. The Newton step is given by

$$\Delta\beta = \left(\sum_i |\Psi^{''}(\gamma_i)| x_i x_i^T\right)^{-1} \sum_i \Psi^{'}(\gamma_i) x_i$$

where all derivatives are evaluated at the current estimate of γ, 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. 420) or when the relative change in the log-likelihood from one iteration to the next is less than `ConvergenceTolerance`/100. Convergence is also assumed after Imsl.Stat.CategoricalGenLinModel.MaxIterations (p. 422) 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 β_m occurs at

$$\hat{\beta_m} = \infty$$

leading to an infinite estimate of both β_m and η_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 η_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 η_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 η 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 γ_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 γ_i when evaluated at the optimal $\hat{\beta}$ and the derivatives here (and only here) are with respect to γ rather than with respect to β. 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

$$\ell_i^{'}(\hat{\gamma}_i)^T \ell^{''}(\hat{\gamma})^{-1} \ell^{'}(\hat{\gamma}_i)$$

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 β.

Programming Notes

1. Classification variables are specified via Imsl.Stat.CategoricalGenLinModel.ClassificationVariableColumn (p. 419). 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. 423) 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. 427) method must be called before using any property as a right operand, otherwise the value is `null`.

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

Covariate(x)	N	y
1.755	62	18
1.784	56	28
1.811	63	52
1.836	59	53
1.861	62	61
1.883	60	60

```
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 = {{1.69, 59.0, 6.0},
                                {1.724, 60.0, 13.0},
                                {1.755, 62.0, 18.0},
                                {1.784, 56.0, 28.0},
                                {1.811, 63.0, 52.0},
                                {1.836, 59.0, 53.0},
                                {1.861, 62.0, 61.0},
                                {1.883, 60.0, 60.0}};

        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

-60.7568  5.1876  -11.7118  0.0000
 34.2985  2.9164   11.7607  0.0000

Log likelihood -18.778179042334
Asymptotic Coefficient Covariance

 26.9117  -15.1243
           8.5052

               Case Analysis

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

 Last Coefficient Update
            0
0  1.85192237772546E-07
1  1.33163785436183E-05
```

```
Covariate Means
     0
0  1.793
1  0

Observation Codes
   0
0  0
1  0
2  0
3  0
4  0
5  0
6  0
7  0

Number of Missing Values 0

MODEL4
Log likelihood -18.2323545743845
        Coefficient Statistics

-34.9441  2.6412  -13.2305  0.0000
 19.7367  1.4852   13.2888  0.0000
```

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

ilt	irt	icen	Class 1	Class 2
0	5	0	1	0
9	4	3	0	0
0	4	1	0	0
9	0	2	1	1
0	1	0	0	1

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.

```
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},
                                {0.0, 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
0  -2.84092901922464E-07
1   3.53822215072981E-10
2   7.09878432577707E-07

Covariate Means
    0
0  0.6
1  0.6
2  0

Distinct Values For Each Class Variable
   0
0  0
1  1
2  0
3  1

Number of Missing Values 0
```

CategoricalGenLinModel.DistributionParameterModel Enumeration

Summary

Indicates the function used to model the distribution parameter.

public enumeration Imsl.Stat.CategoricalGenLinModel.DistributionParameterModel

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
public Imsl.Stat.CategoricalGenLinModel.DistributionParameterModel 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
public Imsl.Stat.CategoricalGenLinModel.DistributionParameterModel 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
public Imsl.Stat.CategoricalGenLinModel.DistributionParameterModel 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

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

Summary

Performs a sign test.

`public class Imsl.Stat.SignTest`

Properties

NumPositiveDev

`public int NumPositiveDev {get; }`

Description

Returns the number of positive differences.

NumZeroDev

`public int NumZeroDev {get; }`

Description

Returns the number of zero differences.

Percentage

`public double Percentage {get; set; }`

Description

The percentage percentile of the population.

`Percentile` is the 100 * `percentage` percentile of the population.

Default: `Percentage` = 0.5.

Percentile

`public double Percentile {get; set; }`

Description

The hypothesized percentile of the population.

Default: `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.

Call this value probability. If using default values, the null hypothesis is that the median equals 0.0.

Returns

A `double` scalar containing the Binomial probability of `NumPositiveDev` or more positive differences in `x.length - number of zero differences` trials.

Description

Class `SignTest` tests hypotheses about the proportion p of a population that lies below a value q, where p and q corresponds 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 $x[j-1] -$ Percentile for $j = 1, 2, \ldots,$ x.length. The binomial probability of the number of values above q in the number of positive differences $x[j-1] -$ Percentile for $j = 1, 2, \ldots, \ldots,$ 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)

  $H_1 : Pr((x \leq q) < p)$ 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 $q = c$.

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.

```
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.

```
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

Summary

Performs a Wilcoxon rank sum test.

`public class Imsl.Stat.WilcoxonRankSum`

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.

The statistics are as follows:

Row	Statistics
0	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
1	2 x E(W) - W, where E(W) is the expected value of W
2	probability of obtaining a statistic less than or equal to min{W, 2 x E(W) - W}
3	W statistic adjusted for ties in such a manner that $W$is as large as possible
4	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
5	probability of obtaining a statistic less than or equal to min{W, 2 x E(W) - W}, adjusted for ties in such a manner that W is as large as possible
6	Wstatistic with average ranks used in case of ties
7	estimated standard error of Row 6 under the null hypothesis of no difference
8	standard normal score associated with Row 6
9	two-sided p-value associated with Row 8

Returns

A `double` array of length 10 containing statistics.

SetFuzz

`public void SetFuzz(double fuzz)`

Description

Sets the nonnegative constant used to determine ties in computing ranks in the combined samples.

A tie is declared when two observations in the combined sample are within `fuzz` of each other. Default: fuzz $= 100 \times 2.2204460492503131e - 16 \times \max(|x_{i1}|, |x_{j2}|)$

Parameter

`fuzz` – A `double` scalar containing the nonnegative constant used to determine ties in computing ranks in the combined samples.

Description

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 `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 `fuzz` 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, `stat[0]` or `stat[3]`,should be used, where `stat` is the array containing the statistics returned from the `getStatistics` method.

Test	Null Hypothesis	Alternative Hypothesis	Action
1	$H_0 : \Pr(x1 < x2) = 0.5$ $H_0 : E(x1) = E(x2)$	$H_1 : \Pr(x1 < x2) \neq 0.5$ $H_1 : E(x1) \neq E(x2)$	Reject if `stat[9]` is less than the significance level of the test. Alternatively, reject the null hypothesis if `stat[6]` is too large or too small.
2	$H_0 : \Pr(x1 < x2) \leq 0.5$ $H_0 : E(x1) \geq E(x2)$	$H_1 : \Pr(x1 < x2) \neq 0.5$ $H_1 : E(x1) < E(x2)$	Reject if `stat[6]` is too small
3	$H_0 : \Pr(x1 < x2) \geq 0.5$ $H_0 : E(x1) \leq E(x2)$	$H_1 : \Pr(x1 < x2) < 0.5$ $H_1 : E(x1) > E(x2)$	Reject if `stat[6]` is too large

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.

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.

```
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

```
p-value = 0.1412
Imsl.Stat.WilcoxonRankSum: "x.length" = 5 and "y.length" = 5.
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.
```

Example 2: Wilcoxon Rank Sum Test

The following example uses the same data as in example 1. Now, all the statistics are displayed.

```
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class WilcoxonRankSumEx2
{
    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};
        String[] labels =new String[]{
                 "Wilcoxon W statistic .....................",
                 "2*E(W) - W ...............................",
                 "p-value ................................. ",
                 "Adjusted Wilcoxon statistic ..............",
                 "Adjusted 2*E(W) - W ......................",
                 "Adjusted p-value ........................ ",
                 "W statistics for averaged ranks...........",
                 "Standard error of W (averaged ranks) ...... ",
                 "Standard normal score of W (averaged ranks) ",
                 "Two-sided p-value of W (averaged ranks) ... "};

        WilcoxonRankSum wilcoxon = new WilcoxonRankSum(x, y);
        wilcoxon.Compute();
        double[] stat = wilcoxon.GetStatistics();

        for (int i = 0; i < 10; i++)
        {
            Console.Out.WriteLine
                (labels[i] + " " + stat[i].ToString("0.000"));
        }
    }
}
```

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
Imsl.Stat.WilcoxonRankSum: "x.length" = 5 and "y.length" = 5.
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

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.

ChiSquaredTest Class

Summary

Chi-squared goodness-of-fit test.

`public class Imsl.Stat.ChiSquaredTest`

Properties

ChiSquared

```
public double ChiSquared {get; }
```

Description

The chi-squared statistic.

DegreesOfFreedom

```
public double DegreesOfFreedom {get; }
```

Description

Returns the degrees of freedom in chi-squared.

P

```
public double P {get; }
```

Description

The p-value for the chi-squared statistic.

Constructors

ChiSquaredTest

```
public ChiSquaredTest(Imsl.Stat.ICdfFunction cdf, double[] cutpoints, int
  nParameters)
```

Description

Constructor for the Chi-squared goodness-of-fit test.

Parameters

`cdf` – Object that implements the `ICdfFunction` interface.

`cutpoints` – A `double` array containing the cutpoints.

`nParameters` – A `int` which specifies the number of parameters estimated in computing the `Cdf`.

`Imsl.Stat.NotCDFException` id is thrown if the function `cdf.CdfFunction` is not a valid CDF.

ChiSquaredTest

```
public ChiSquaredTest(Imsl.Stat.ICdfFunction cdf, int nCutpoints, int
  nParameters)
```

Description

Constructor for the Chi-squared goodness-of-fit test

Parameters

`cdf` – Object that implements the `ICdfFunction` interface.

`nCutpoints` – A `int` which specifies the number of cutpoints.

`nParameters` – A `int` which specifies the number of parameters estimated in computing the `Cdf`.

`Imsl.Stat.NotCDFException` id is thrown if the function `cdf.CdfFunction` is not a valid CDF.

`Imsl.Stat.DidNotConvergeException` id is thrown if the interation to find the inverse of the CDF did not converge. The inverse CDF is needed to compute the cutpoints.

Methods

GetCellCounts

`public double[] GetCellCounts()`

Description

Returns the cell counts.

Returns

A `double` array which contains the number of actual observations in each cell.

GetCutpoints

`public double[] GetCutpoints()`

Description

Returns the cutpoints.

The intervals defined by the cutpoints are such that the lower endpoint is not included while the upper endpoint is included in the interval.

Returns

A `double` array which contains the cutpoints.

GetExpectedCounts

`public double[] GetExpectedCounts()`

Description

Returns the expected counts.

Returns

A `double` array which contains the number of expected observations in each cell.

SetCutpoints

`public void SetCutpoints(double[] cutpoints)`

Description

Sets the cutpoints.

The intervals defined by the cutpoints are such that the lower endpoint is not included while the upper endpoint is included in the interval.

Parameter

`cutpoints` – A `double` array which contains the cutpoints.

SetRange

`public void SetRange(double lower, double upper)`

Description

Sets endpoints of the range of the distribution.

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.

Parameters

`lower` – A `double` which specifies the lower range limit.

`upper` – A `double` which specifies the upper range limit.

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.

Description

`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.

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
        Imsl.Stat.Random rn = new Imsl.Stat.Random(123457);
        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 4.79629666357385
The P-value is 0.441242957205531
The Degrees of freedom are 5
Imsl.Stat.ChiSquaredTest: An expected value is less than five.
```

NormalityTest Class

Summary

Performs a test for normality.

```
public class Imsl.Stat.NormalityTest
```

Properties

ChiSquared

`public double ChiSquared {get; }`

Description

Returns the chi-square statistic for the chi-squared goodness-of-fit test.

Returns `Double.NaN` for other tests.

DegreesOfFreedom

`public double DegreesOfFreedom {get; }`

Description

Returns the degrees of freedom for the chi-squared goodness-of-fit test.

Returns `Double.NaN` for other tests.

MaxDifference

`public double MaxDifference {get; }`

Description

Returns the maximum absolute difference between the empirical and the theoretical distributions for the Lilliefors test.

Returns `Double.NaN` for other tests.

ShapiroWilkW

`public double ShapiroWilkW {get; }`

Description

Returns the Shapiro-Wilk W statistic for the Shapiro-Wilk W test.

Returns `Double.NaN` for other tests.

Constructor

NormalityTest

`public NormalityTest(double[] x)`

Description

Constructor for `NormalityTest`.

`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.

Parameter

`x` – A `double` array containing the observations.

Methods

ChiSquaredTest

`public double ChiSquaredTest(int n)`

Description

Performs the chi-squared goodness-of-fit test.

See Also: Imsl.Stat.NormalityTest.ChiSquaredTest(System.Int32) (p. 458)

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.

`Imsl.Stat.NoVariationInputException` id is thrown if there is no variation in the input data

`Imsl.Stat.DidNotConvergeException` id is thrown if the iteration did not converge

LillieforsTest

`public double LillieforsTest()`

Description

Performs the Lilliefors test.

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.

Returns

A `double` scalar containing the p-value for the Lilliefors test.

`Imsl.Stat.NoVariationInputException` id is thrown if there is no variation in the input data

`Imsl.Stat.DidNotConvergeException` id 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.

`Imsl.Stat.NoVariationInputException` id is thrown if there is no variation in the input data

`Imsl.Stat.DidNotConvergeException` id is thrown if the iteration did not converge

Description

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

$$W = \left(\sum a_i x_{(i)}\right)^2 / \left(\sum (x_i - \bar{x})^2\right)$$

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`.

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,
                                     77.0, 31.0, 43.0, 58.0, 65.0, 81.0,
                                     32.0, 44.0, 58.0, 66.0, 87.0, 33.0,
                                     45.0, 58.0, 68.0, 89.0, 33.0, 48.0,
                                     58.0, 68.0, 93.0, 35.0, 48.0, 59.0,
                                     70.0, 97.0};

        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
```

Chapter 18: Time Series and Forecasting

Types

Usage Notes

The classes in this chapter assume the time series does not contain any missing observations. If missing values are present, they should be set to NaN (see `Double.NaN`), and the classes will return an appropriate error message. To enable fitting of the model, the missing values must be replaced by appropriate estimates.

General Methodology

A major component of the model identification step concerns determining if a given time series is stationary. The sample correlation functions computed by the AutoCorrelation class methods `getAutoCorrelations` and `getPartialAutoCorrelations` may be used to diagnose the presence of nonstationarity in the data, as well as to indicate the type of transformation required to induce stationarity.

The "raw" data and sample correlation functions provide insight into the nature of the underlying model. Typically, this information is displayed in graphical form via time series plots, plots of the lagged data, and various correlation function plots.

ARIMA Model (Autoregressive Integrated Moving Average)

A small, yet comprehensive, class of stationary time-series models consists of the nonseasonal ARMA processes defined by

$$\phi(B)(W_t - \mu) = \theta(B)A_t, \qquad t \in Z$$

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:

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p, p \geq 0$$

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q, q \geq 0$$

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

$$\phi(B)W_t = \theta_0 + \theta(B)A_i, \quad t \in Z$$

where θ_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.

Typically, the method of moments includes setting property `Method` to `MethodOfMoments` in the ARMA class for preliminary parameter stimates. These estimates can be used as initial values into the least-squares procedure by setting property `Method` to `LeastSquares` in the ARMA class. 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 `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).

AutoCorrelation Class

Summary

Computes the sample autocorrelation function of a stationary time series.

`public class Imsl.Stat.AutoCorrelation`

Properties

Mean

`public double Mean {get; set; }`

Description

The mean of the time series `x`.

Variance

`public double Variance {get; }`

Description

Returns 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.

`maximumLag` must be greater than or equal to 1 and less than the number of observations in `x`.

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.

Methods

GetAutoCorrelations

```
public double[] GetAutoCorrelations()
```

Description

Returns the autocorrelations of the time series `x`.

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`.

Returns

A `double` array of length `maximumLag` +1 containing the autocorrelations of the time series `x`.

GetAutoCovariances

```
public double[] GetAutoCovariances()
```

Description

Returns the variance and autocovariances of the time series `x`.

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`.

Returns

A `double` array of length `maximumLag` + 1 containing the variances and autocovariances of the time series `x`.

`Imsl.Stat.NonPosVarianceException` id 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

```
public double[] GetStandardErrors(Imsl.Stat.AutoCorrelation.StdErr
  stderrMethod)
```

Description

Returns the standard errors of the autocorrelations of the time series `x`.

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 `Morans`, Moran's formula is used to compute the standard errors of autocorrelations.

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`.

Description

`AutoCorrelation` estimates the autocorrelation function of a stationary time series given a sample of n observations $\{X_t\}$ for $\mathrm{t} = 1, 2, \ldots, \mathrm{n}$.

Let

$$\hat{\mu} = \mathrm{xmean}$$

be the estimate of the mean μ of the time series $\{X_t\}$ where

$$\hat{\mu} = \begin{cases} \mu & \text{for } \mu \text{ known} \\ pa \quad \frac{1}{n}\sum\limits_{t=1}^{n} X_t & \text{for } \mu \text{ unknown} \end{cases}$$

The autocovariance function $\sigma(k)$ is estimated by

$$\hat{\sigma}\left(k\right) = \frac{1}{n}\sum_{t=1}^{n-k}\left(X_t - \hat{\mu}\right)\left(X_{t+k} - \hat{\mu}\right), \qquad \text{k=0,1,\ldots,K}$$

where $K =$ `maximumLag`. Note that $\hat{\sigma}(0)$ is an estimate of the sample variance. The autocorrelation function $\rho(k)$ is estimated by

$$\hat{\rho}(k) = \frac{\hat{\sigma}(k)}{\hat{\sigma}(0)}, \qquad k = 0, 1, \ldots, K$$

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

$$\text{var}\{\hat{\rho}(k)\} = \frac{1}{n}\sum_{i=-\infty}^{\infty}\left[\rho^2(i) + \rho(i-k)\rho(i+k) - 4\rho(i)\rho(k)\rho(i-k) + 2\rho^2(i)\rho^2(k)\right]$$

where $\hat{\rho}(k)$ assumes μ is unknown. For computational purposes, the autocorrelations $\rho(k)$ are replaced by their estimates $\hat{\rho}(k)$ for $|k| \le K$, 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

$$var\{\hat{\rho}(k)\} = \frac{n-k}{n(n+2)}$$

where μ 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 = `maximumLag` sample autocorrelations $\hat{\rho}(k)$ for k=0,1,...,K. Consider the AR(k) process defined by

$$X_t = \phi_{k1}X_{t-1} + \phi_{k2}X_{t-2} + \cdots + \phi_{kk}X_{t-k} + A_t$$

where ϕ_{kj} denotes the j-th coefficient in the process. The set of estimates $\{\hat{\phi}_{kk}\}$ for $k = 1$, ..., K is the sample partial autocorrelation function. The autoregressive parameters $\{\hat{\phi}_{kj}\}$ for $j = 1$, ..., k are approximated by Yule-Walker estimates for successive AR(k) models where $k = 1$, ..., K. Based on the sample Yule-Walker equations

$$\hat{\rho}(j) = \hat{\phi}_{k1}\hat{\rho}(j-1) + \hat{\phi}_{k2}\hat{\rho}(j-2) + \cdots + \hat{\phi}_{kk}\hat{\rho}(j-k), \qquad \text{j = 1,2,\ldots,k}$$

a recursive relationship for k=1, ..., K was developed by Durbin (1960). The equations are given by

$$\hat{\phi}_{kk} = \begin{cases} \hat{\rho}(1) & \text{for k = 1} \\ \dfrac{\hat{\rho}(k) - \sum_{j=1}^{k-1}\hat{\phi}_{k-1,j}\hat{\rho}(k-j)}{1 - \sum_{j=1}^{k-1}\hat{\phi}_{k-1,j}\hat{\rho}(j)} & \text{for k = 2, } \ldots \text{, K} \end{cases}$$

and

$$\hat{\phi}_{kj} = \begin{cases} \hat{\phi}_{k-1,j} - \hat{\phi}_{kk}\hat{\phi}_{k-1,k-j} & \text{for j} = 1, 2, \ldots, \text{k} - 1 \\ \hat{\phi}_{kk} & \text{for j = k} \end{cases}$$

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 $\{\phi_{kk}\}$ 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

$$\mathrm{var}\{\hat{\phi}_{kk}\} \simeq \frac{1}{n} \quad \mathrm{k} \ \geq \ \mathrm{p}+1$$

See Box and Jenkins (1976, pages 82-84) for more information concerning the partial autocorrelation function.

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.

```
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,
                                     95.7, 77.2, 59.1, 44, 47,
                                     30.5, 16.3, 7.3, 37.3, 73.9};

        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("Mean = " + ac.Mean);
        Console.Out.WriteLine();
        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
 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
13   197.75742912
14    26.89107936
15   -77.2807224
```

```
16  -143.73279616
17  -202.04799792
18  -245.37223168
19  -230.81567344
20  -142.8788232


 AutoCorrelations are:
             0
 0   1
 1   0.806293065691258
 2   0.428086653780237
 3   0.0689108813212006
 4  -0.170620023128885
 5  -0.267559955093586
 6  -0.212780177317129
 7  -0.043706718936501
 8   0.164604293249802
 9   0.331461500117813
10   0.410613524938228
11   0.394908424249623
12   0.288477093166393
13   0.143001143740562
14   0.0194453129877855
15  -0.0558827637549379
16  -0.10393518127421
17  -0.146103713633525
18  -0.17743206881559
19  -0.166906019369514
20  -0.103317661565611

Mean = 46.976

Standard Error using Bartlett are:
            0
 0  0.0347838253702384
 1  0.0962419914340011
 2  0.156783378574532
 3  0.205766777086907
 4  0.230955675779118
 5  0.228994712235613
 6  0.208621905639667
 7  0.178475936561125
 8  0.145727084432033
 9  0.134405581638002
10  0.150675803916788
11  0.174348147103935
12  0.190619474429408
13  0.195490061669564
14  0.195892530944597
15  0.196285328179458
16  0.196020624500033
17  0.198716030900604
18  0.205358590947539
19  0.2093868822353
```

```
Standard Error using Moran are:
            0
 0  0.0985184366143778
 1  0.0980196058819607
 2  0.0975182235357506
 3  0.0970142500145332
 4  0.0965076447241154
 5  0.0959983659991659
 6  0.0954863710632231
 7  0.0949716159867634
 8  0.094454055643212
 9  0.0939336436627724
10  0.0934103323839415
11  0.0928840728025648
12  0.0923548145182799
13  0.0918225056781811
14  0.0912870929175277
15  0.090748521297303
16  0.0902067342384192
17  0.0896616734523426
18  0.0891132788679007
19  0.0885614885540095


Partial AutoCovariances:
             0
 0   0.806293065691258
 1  -0.634544877310468
 2   0.0782508772709519
 3  -0.0585660846582815
 4  -0.00094221571933657
 5   0.171719898229681
 6   0.108591873581717
 7   0.11000138764865
 8   0.0785374339029981
 9   0.0791563332964613
10   0.0687065876031485
11  -0.0378019674610775
12   0.0811184838397538
13   0.0334124214991749
14  -0.0348467839607946
15  -0.130648157884444
16  -0.154900984829049
17  -0.119085063160732
18  -0.0161889037437313
19  -0.00385175459253345

AutoCovariances are:
        0
 0  1392.0526
 1  1126.5241
 2   604.1624
 3   106.7545
 4  -225.882
 5  -361.0259
```

```
 6  -286.5701
 7   -53.7603
 8   235.9665
 9   470.7857
10   584.0143
11   564.7639
12   418.3631
13   216.1044
14    43.125
15   -63.4683
16  -131.5012
17  -189.0627
18  -229.6888
19  -212.1559
20  -121.5693


 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   0.419534649768263
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  -0.152405088715757
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
 9  0.133747601649498
10  0.148150190923942
11  0.172282351100035
```

```
12  0.190275929042947
13  0.196791614240352
14  0.197983743593071
15  0.198474748794747
16  0.198318159677368
17  0.201022833791806
18  0.207071652966429
19  0.210217650328868
```

AutoCorrelation.StdErr Enumeration

Summary

Standard Error computation method.

`public enumeration Imsl.Stat.AutoCorrelation.StdErr`

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

Summary

Computes the sample cross-correlation function of two stationary time series.

`public class Imsl.Stat.CrossCorrelation`

Properties

MeanX

`public double MeanX {get; set; }`

Description

Estimate of the mean of time series `x`.

MeanY

`public double MeanY {get; set; }`

Description

Estimate of the mean of time series `y`.

VarianceX

`public double VarianceX {get; }`

Description

Returns the variance of time series `x`.

VarianceY

`public double VarianceY {get; }`

Description

Returns the variance of time series `y`.

Constructor

CrossCorrelation

`public CrossCorrelation(double[] x, double[] y, int maximumLag)`

Description

Constructor to compute the sample cross-correlation function of two stationary time series.

`maximumLag` must be greater than or equal to 1 and less than the minimum of the number of observations of `x` and `y`.

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.

Methods

GetAutoCorrelationX

`public double[] GetAutoCorrelationX()`

Description

Returns the autocorrelations of the time series x.

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`.

Returns

A `double` array of length `maximumLag` + 1 containing the autocorrelations of the time series `x`.

`Imsl.Stat.NonPosVarianceException` id is thrown if the problem is ill-conditioned.

GetAutoCorrelationY

`public double[] GetAutoCorrelationY()`

Description

Returns the autocorrelations of the time series y.

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`.

Returns

A `double` array of length `maximumLag` + 1 containing the autocorrelations of the time series `y`.

`Imsl.Stat.NonPosVarianceException` id is thrown if the problem is ill-conditioned.

GetAutoCovarianceX

`public double[] GetAutoCovarianceX()`

Description

Returns the autocovariances of the time series `x`.

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`.

Returns

A `double` array of length `maximumLag` + 1 containing the variances and autocovariances of the time series `x`.

`Imsl.Stat.NonPosVarianceException` id is thrown if the problem is ill-conditioned.

GetAutoCovarianceY

`public double[] GetAutoCovarianceY()`

Description

Returns the autocovariances of the time series `y`.

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`.

Returns

A `double` array of length `maximumLag` + 1 containing the variances and autocovariances of the time series `y`.

`Imsl.Stat.NonPosVarianceException` id is thrown if the problem is ill-conditioned.

GetCrossCorrelations

```
public double[] GetCrossCorrelations()
```

Description

Returns the cross-correlations between the time series `x` and `y`.

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)`.

Returns

A `double` array of length 2 * `maximumLag` + 1 containing the cross-correlations between the time series `x` and `y`.

`Imsl.Stat.NonPosVarianceXYException` id 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`.

The cross-covariance between `x` and `y` at lag k, where $k =$ `-maximumLag`,..., 0, 1,...,`maximumLag`, corresponds to output array indices `0, 1,..., (2*maximumLag)`.

Returns

A `double` array of length 2 * `maximumLag` + 1 containing the cross-covariances between the time series `x` and `y`.

GetStandardErrors

```
public double[] GetStandardErrors(Imsl.Stat.CrossCorrelation.StdErr
  stderrMethod)
```

Description

Returns the standard errors of the cross-correlations between the time series `x` and `y`.

The standard error of cross-correlations between `x` and `y` at lag k, where $k =$ `-maximumLag`,..., 0, 1,..., `maximumLag`, corresponds to output array indices 0, 1,..., `(2*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.

Parameter

`stderrMethod` – An `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 * `maximumLag` + 1 containing the standard errors of the cross-correlations between the time series `x` and `y`.

`Imsl.Stat.NonPosVarianceException` id is thrown if the problem is ill-conditioned.

Description

`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, ..., n.

Let

$$\hat{\mu}_x = \text{xmean}$$

be the estimate of the mean μ_X of the time series $\{X_t\}$ where

$$\hat{\mu}_X = \begin{cases} \mu_X & \text{for } \mu_X \text{ known} \\ \frac{1}{n}\sum\limits_{t=1}^{n} X_t & \text{for } \mu_X \text{ unknown} \end{cases}$$

The autocovariance function of $\{X_t\}$, $\sigma_X(k)$, is estimated by

$$\hat{\sigma}_X(k) = \frac{1}{n}\sum_{t=1}^{n-k}\left(X_t - \hat{\mu}_X\right)\left(X_{t+k} - \hat{\mu}_X\right), \qquad \text{k=0,1,\ldots,K}$$

where K = `maximumLag`. Note that $\hat{\sigma}_X(0)$ is equivalent to the sample variance of `x` returned by property `VarianceX`. The autocorrelation function $\rho_X(k)$ is estimated by

$$\hat{\rho}_X(k) = \frac{\hat{\sigma}_X(k)}{\hat{\sigma}_X(0)}, \qquad k = 0, 1, \ldots, K$$

Note that $\hat{\rho}_x(0) \equiv 1$ 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

$$\hat{\sigma}_{XY}(k) = \begin{cases} \frac{1}{n}\sum\limits_{t=1}^{n-k}(X_t - \hat{\mu}_X)(Y_{t+k} - \hat{\mu}_Y) & k = 0, 1, \ldots, K \\ \frac{1}{n}\sum\limits_{t=1-k}^{n}(X_t - \hat{\mu}_X)(Y_{t+k} - \hat{\mu}_Y) & k = -1, -2, \ldots, -K \end{cases}$$

The cross-correlation function $\rho_{XY}(k)$ is estimated by

$$\hat{\rho}_{XY}(k) = \frac{\hat{\sigma}_{XY}(k)}{[\hat{\sigma}_X(0)\hat{\sigma}_Y(0)]^{\frac{1}{2}}} \quad k = 0, \pm 1, \ldots, \pm K$$

The standard errors of the sample cross-correlations may be optionally computed according to the `GetStandardErrors` method argument `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

$$\begin{aligned} \text{var}\{\hat{\rho}_{XY}(k)\} = \tfrac{1}{n-k}\sum_{i=-\infty}^{\infty} [\ & \rho_X(i) + \rho_{XY}(i-k)\rho_{XY}(i+k) \\ & -2\rho_{XY}(k)\{\rho_X(i)\rho_{XY}(i+k) + \rho_{XY}(-i)\rho_Y(i+k)\} \\ & +\rho_{XY}^2(k)\{\rho_X(i) + \tfrac{1}{2}\rho_X^2(i) + \tfrac{1}{2}\rho_Y^2(i)\}\ ] \end{aligned}$$

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.

Example 1: CrossCorrelation

Consider the Gas Furnace Data (Box and Jenkins 1976, pages 532-533) where X is the input gas reate 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 -maximumLag = -10 through lag 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 maximumLag or less than -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, 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};
        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,
                                          - 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,
56.0, 55.4, 55.4, 56.4, 57.2,
58.0, 58.4, 58.4, 58.1, 57.7,
57.0, 56.0, 54.7, 53.2, 52.1,
51.6, 51.0, 50.5, 50.4, 51.0,
51.8, 52.4, 53.0, 53.4, 53.6,
53.7, 53.8, 53.8, 53.8, 53.3,
53.0, 52.9, 53.4, 54.6, 56.4,
58.0, 59.4, 60.2, 60.0, 59.4,
58.4, 57.6, 56.9, 56.4, 56.0,
55.7, 55.3, 55.0, 54.4, 53.7,
52.8, 51.6, 50.6, 49.4, 48.8,
48.5, 48.7, 49.2, 49.8, 50.4,
50.7, 50.9, 50.7, 50.5, 50.4,
50.2, 50.4, 51.2, 52.3, 53.2,
53.9, 54.1, 54.0, 53.6, 53.2,
53.0, 52.8, 52.3, 51.9, 51.6,
51.6, 51.4, 51.2, 50.7, 50.0,
49.4, 49.3, 49.7, 50.6, 51.8,
53.0, 54.0, 55.3, 55.9, 55.9,
54.6, 53.5, 52.4, 52.1, 52.3,
53.0, 53.8, 54.6, 55.4, 55.9,
55.9, 55.2, 54.4, 53.7, 53.6,
53.6, 53.2, 52.5, 52.0, 51.4,
51.0, 50.9, 52.4, 53.5, 55.6,
58.0, 59.5, 60.0, 60.4, 60.5,
60.2, 59.7, 59.0, 57.6, 56.4,
55.2, 54.5, 54.1, 54.1, 54.4,
55.5, 56.2, 57.0, 57.3, 57.4,
57.0, 56.4, 55.9, 55.5, 55.3,
55.2, 55.4, 56.0, 56.5, 57.1,
57.3, 56.8, 55.6, 55.0, 54.1,
54.3, 55.3, 56.4, 57.2, 57.8,
58.3, 58.6, 58.8, 58.8, 58.6,
58.0, 57.4, 57.0, 56.4, 56.3,
56.4, 56.4, 56.0, 55.2, 54.0,
53.0, 52.0, 51.6, 51.6, 51.1,
50.4, 50.0, 50.0, 52.0, 54.0,
55.1, 54.5, 52.8, 51.4, 50.8,
51.2, 52.0, 52.8, 53.8, 54.5,
54.9, 54.9, 54.8, 54.4, 53.7,
53.3, 52.8, 52.6, 52.6, 53.0,
54.3, 56.0, 57.0, 58.0, 58.6,
58.5, 58.3, 57.8, 57.3, 57.0};
```

```
        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  -1.65852650999817
11  -2.04865124574232
12  -2.48216585776478
13  -2.88541054192018
```

```
14  -3.16536049680239
15  -3.25343758942199
16  -3.13112860301494
17  -2.83919398544463
18  -2.45302186901565
19  -2.05268794195849
20  -1.6946546517713

CrossCorrelations are:
            0
 0  -0.118153717307789
 1  -0.148528662561878
 2  -0.179455515102209
 3  -0.206067503381416
 4  -0.226715971265165
 5  -0.242870996488244
 6  -0.260350586329711
 7  -0.286431898500946
 8  -0.3285421835153
 9  -0.39346731487308
10  -0.484450717109386
11  -0.598405005361053
12  -0.725033348897091
13  -0.842819935503927
14  -0.924592494205792
15  -0.950319553992448
16  -0.914593458680361
17  -0.829320215245049
18  -0.716520475473708
19  -0.599584112456951
20  -0.495003641096017

Standard Errors using Bartlett are:
            0
 0  0.158147783754555
 1  0.155750271182418
 2  0.152735096430409
 3  0.149086745416716
 4  0.145054998300008
 5  0.141300099196058
 6  0.138420534019813
 7  0.136074039397204
 8  0.132158917844376
 9  0.123531347020305
10  0.107879045104545
11  0.0873410658167485
12  0.0641407975847026
13  0.0469456102701398
14  0.0440970262220149
15  0.0482335854893665
16  0.0491545707033738
17  0.0475621871011123
18  0.0534780426550682
19  0.0715660938138719
20  0.0939330263600716
```

```
Standard Errors using Bartlett #2 are:
             0
 0  0.162753654681801
 1  0.162469864309526
 2  0.162187553298139
 3  0.161906708839297
 4  0.161627318279375
 5  0.161349369117073
 6  0.16107284900106
 7  0.160797745727675
 8  0.160524047238664
 9  0.160251741618955
10  0.159980817094486
11  0.160251741618955
12  0.160524047238664
13  0.160797745727675
14  0.16107284900106
15  0.161349369117073
16  0.161627318279375
17  0.161906708839297
18  0.162187553298139
19  0.162469864309526
20  0.162753654681801

AutoCovariances of X are:
             0
 0  1.14693790165038
 1  1.09242958215267
 2  0.956651878489968
 3  0.782050821478561
 4  0.609290776371371
 5  0.467379623909361
 6  0.36495658921123
 7  0.298426970727032
 8  0.260942845999682
 9  0.244377603086156
10  0.238942463361545

AutoCovariances of Y are:
             0
 0  10.2189370662893
 1   9.92010118439122
 2   9.15657243817617
 3   8.09900196442277
 4   6.94850770962479
 5   5.87055032023953
 6   4.96076244327211
 7   4.25188969596136
 8   3.73611877936647
 9   3.37615547781905
10   3.13231605775447

AutoCorrelations of X are:
             0
 0  1
 1  0.952474916541448
```

```
 2  0.834092130980584
 3  0.681859776674248
 4  0.531232576318765
 5  0.407502117801518
 6  0.31820082733867
 7  0.26019453215175
 8  0.227512619143721
 9  0.213069602752258
10  0.208330776250152

AutoCorrelations of Y are:
            0
 0  1
 1  0.970756656983059
 2  0.896039615351222
 3  0.79254837483442
 4  0.67996384208558
 5  0.574477588242088
 6  0.485447988483746
 7  0.416079448222429
 8  0.365607377277169
 9  0.330382255602345
10  0.306520730819207
```

CrossCorrelation.StdErr Enumeration

Summary

Standard Error computation method.

`public enumeration Imsl.Stat.CrossCorrelation.StdErr`

Fields

`Bartletts`
`public Imsl.Stat.CrossCorrelation.StdErr 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

Summary

Computes the multichannel cross-correlation function of two mutually stationary multichannel time series.

`public class Imsl.Stat.MultiCrossCorrelation`

Constructor

MultiCrossCorrelation

`public MultiCrossCorrelation(double[,] x, double[,] y, int maximumLag)`

Description

Constructor to compute the multichannel cross-correlation function of two mutually stationary mulitchannel 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

`public double[,,] GetCrossCorrelation()`

Description

Returns the cross-correlations between the channels of `x` and `y`.

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)`.

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`.

`Imsl.Stat.NonPosVarianceXYException` id 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`.

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)`.

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`.

`Imsl.Stat.NonPosVarianceXYException` id 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`.

`Imsl.Stat.NonPosVarianceXYException` id 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`.

`Imsl.Stat.NonPosVarianceXYException` id is thrown if the problem is ill-conditioned. The variance is too small to work with.

Description

`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 $\hat{\mu}_X =$ `xmean` 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 $j = 1, 2, ..., p$

$$\hat{\mu}_{X_j} = \begin{cases} \mu_{X_j} & \text{for } \mu_{X_j} \text{ known} \\ \frac{1}{n}\sum_{t=1}^{n} X_{tj} & \text{for } \mu_{X_j} \text{ unknown} \end{cases}$$

Let $\hat{\mu}_Y$ = `ymean` be similarly defined. The cross-covariance of lag k between channel i of X and channel j of Y is estimated by

$$\hat{\sigma}_{X_iY_j}(k) = \begin{cases} \frac{1}{N}\sum_t (X_{ti} - \hat{\mu}_{X_i})(Y_{t+k,j} - \hat{\mu}_{Y_j}) & k = 0, 1, \ldots, K \\ \frac{1}{N}\sum_t (X_{ti} - \hat{\mu}_{X_i})(Y_{t+k,j} - \hat{\mu}_{Y_j}) & k = -1, -2, \ldots, -K \end{cases}$$

where $i = 1, ..., p$, $j = 1, ..., q$, and K = `maximumLag`. The summation on t extends over all possible cross-products with N equal to the number of cross-products in the sum.

Let $\hat{\sigma}_X(0)$ = `xvar`, where `xvar` 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

$$\hat{\sigma}_{X_j}(0) = \frac{1}{n}\sum_{t=1}^{n}\left(X_{tj} - \hat{\mu}_{X_j}\right)^2, \qquad \text{j=0,1,\ldots,p}$$

Let $\hat{\sigma}_Y(0)$ = `yvar`, where `yvar` 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

$$\hat{\rho}_{X_jY_j}(k) = \frac{\hat{\sigma}_{X_jY_j(k)}}{[\hat{\sigma}_{X_i}(0)\hat{\sigma}_{X_j}(0)]^{\frac{1}{2}}} \qquad k = 0, \pm 1, \ldots, \pm K$$

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 -`maximumLag` = -10 through lag `maximumLag` = 10.

```
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},
    {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},
    {87.0, 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}};

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},
                    {10.0}, {24.0},
                    {83.0}, {132.0},
                    {131.0}, {118.0},
                    {90.0}, {67.0},
                    {60.0}, {47.0},
                    {41.0}, {21.0},
                    {16.0}, {6.0},
                    {4.0}, {7.0},
                    {14.0}, {34.0},
                    {45.0}, {43.0},
                    {48.0}, {42.0},
                    {28.0}, {10.0},
                    {8.0}, {2.0},
                    {0.0}, {1.0},
                    {5.0}, {12.0},
                    {14.0}, {35.0},
                    {46.0}, {41.0},
                    {30.0}, {24.0},
                    {16.0}, {7.0},
                    {4.0}, {2.0},
                    {8.0}, {17.0},
                    {36.0}, {50.0},
                    {62.0}, {67.0},
                    {71.0}, {48.0},
                    {28.0}, {8.0},
                    {13.0}, {57.0},
                    {122.0}, {138.0},
                    {103.0}, {86.0},
                    {63.0}, {37.0},
                    {24.0}, {11.0},
                    {15.0}, {40.0},
                    {62.0}, {98.0},
                    {124.0}, {96.0},
                    {66.0}, {64.0},
                    {54.0}, {39.0},
                    {21.0}, {7.0},
                    {4.0}, {23.0},
                    {55.0}, {94.0},
                    {96.0}, {77.0},
                    {59.0}, {44.0},
                    {47.0}, {30.0},
                    {16.0}, {7.0},
                    {37.0}, {74.0}};

        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

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
0   922.759311111112
1  -279.286422222222

Multichannel cross-correlation between X and Y
Lag K = -10
  CrossCorrelations :
            0
0  -0.0107245938219684
1   0.0427376557935899

Lag K = -9
 CrossCorrelations :
           0
0  0.0339970370656115
1  0.023048812287829

Lag K = -8
 CrossCorrelations :
           0
0  0.113265706453004
1  0.0819407975561327

Lag K = -7
 CrossCorrelations :
           0
0  0.129032618058936
1  0.0606569035081169

Lag K = -6
 CrossCorrelations :
           0
0  0.074309566502109
1  0.0271776680765982

Lag K = -5
  CrossCorrelations :
            0
0   0.0265062285548632
1  -0.00713740085770933

Lag K = -4
 CrossCorrelations :
           0
0  0.0380021196855836
1  0.0197587668528454

Lag K = -3
  CrossCorrelations :
            0
0   0.11390192098873
```

```
1  -0.0242468161934945

Lag K = -2
  CrossCorrelations :
            0
0   0.186035762912295
1  -0.0922580420292281

Lag K = -1
 CrossCorrelations :
           0
0   0.303063597562697
1  -0.128702809263875

Lag K = 0
  CrossCorrelations :
            0
0   0.429575382251174
1  -0.0633098708358119

Lag K = 1
 CrossCorrelations :
           0
0  0.423565683647071
1  0.0333377002981115

Lag K = 2
 CrossCorrelations :
           0
0  0.328542235922487
1  0.051236397797642

Lag K = 3
 CrossCorrelations :
           0
0  0.22914425606054
1  0.0459105243818767

Lag K = 4
 CrossCorrelations :
           0
0  0.124849394067548
1  0.121107717407232

Lag K = 5
 CrossCorrelations :
           0
0  0.075090277447643
1  0.171031279954621

Lag K = 6
 CrossCorrelations :
           0
0  0.132263745693782
1  0.141662566889261
```

```
Lag K = 7
 CrossCorrelations :
          0
0  0.250683184784367
1  0.134797082107539

Lag K = 8
 CrossCorrelations :
          0
0  0.37901007257894
1  0.0752190432013873

Lag K = 9
  CrossCorrelations :
           0
0   0.48360807434863
1  -0.0480587280714567

Lag K = 10
 CrossCorrelations :
          0
0   0.48245160241607
1  -0.168795069078383
```

ARMA Class

Summary

Computes least-square estimates of parameters for an ARMA model.

`public class Imsl.Stat.ARMA`

Properties

BackwardOrigin

`public int BackwardOrigin {get; set; }`

Description

The maximum backward origin.

`BackwardOrigin` must be greater than or equal to 0 and less than or equal to `z.Length - Math.max(maxar, maxma)`, where

`maxar = Math.max(ARLags[i])`, `maxma = Math.max(MALags[j])`, and forecasts at origins `z.Length - BackwardOrigin` through `z.Length` are generated. Default: `BackwardOrigin` = 0.

Center

```
public bool Center {get; set; }
```

Description

The center option.

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 percent probability limits of the forecasts.

Typical choices for `Confidence` are 0.90, 0.95, and 0.99. `Confidence` must be greater than 0.0 and less than 1.0. Default: `Confidence` = 0.95.

Constant

```
public double Constant {get; }
```

Description

Returns the constant parameter estimate.

ConvergenceTolerance

```
public double ConvergenceTolerance {get; set; }
```

Description

The tolerance level used to determine convergence of the nonlinear least-squares algorithm.

`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. The default value is $\max(10^{-20}, \text{eps}^{2/3})$, where `eps` = 2.2204460492503131e-16.

MaxIterations

```
public int MaxIterations {get; set; }
```

Description

The maximum number of iterations.

Default: `MaxIterations` = 200.

MeanEstimate

```
public double MeanEstimate {get; set; }
```

Description

An update of the mean of the time series z.

If the time series is not centered about its mean, and least-squares algorithm is used, the mean is not used in parameter estimation.

Method

`public Imsl.Stat.ARMA.ParamEstimation Method {get; set; }`

Description

The method used to estimate the autoregressive and moving average parameters estimates.

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`.

RelativeError

`public double RelativeError {get; set; }`

Description

The stopping criterion for use in the nonlinear equation solver.

Default: `RelativeError` = 100 * 2.2204460492503131e-16.

SSResidual

`public double SSResidual {get; }`

Description

Returns the sum of squares of the random shock.

This property is only applicable using least-squares algorithm.

Variance

`public double Variance {get; }`

Description

Returns the variance of the time series `z`.

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.

`System.ArgumentException` id 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.

`Imsl.Stat.MatrixSingularException` id is thrown if the input matrix is singular

`Imsl.Stat.TooManyCallsException` id is thrown if the number of calls to the function has exceeded

`Imsl.Stat.IncreaseErrRelException` id is thrown if the bound for the relative error is too small

`Imsl.Stat.NewInitialGuessException` id is thrown if the iteration has not made good progress

`Imsl.Stat.IllConditionedException` id is thrown if the problem is ill-conditioned

`Imsl.Stat.TooManyIterationsException` id is thrown if the maximum number of iterations exceeded

`Imsl.Stat.TooManyFunctionEvaluationsException` id is thrown if the maximum number of function evaluations exceeded

`Imsl.Stat.TooManyJacobianEvalException` id is thrown if the maximum number of Jacobian evaluations exceeded

Forecast

`public double[,] Forecast(int nPredict)`

Description

Computes forecasts and their associated probability limits for an ARMA model.

Parameter

`nPredict` – An `int` scalar containing the maximum lead time for forecasts. `nPredict` must be greater than 0.

Returns

A `double` matrix of dimensions of `nPredict` 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.

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, ..., p + q + 1`.

GetDeviations

`public double[] GetDeviations()`

Description

Returns the deviations from each forecast that give the confidence percent probability limits.

Returns

A `double` array of length `nPredict` containing the deviations from each forecast that give the confidence percent probability limits.

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.

GetParamEstimatesCovariance

`public double[,] GetParamEstimatesCovariance()`

Description

Returns the covariances of parameter estimates.

The ordering of variables is `mean`, `AR`, and `MA`.

Returns

A `double` matrix of `np` by `np` dimensions, where `np = p + q + 1` if `z` is centered about `MeanEstimate`, and `np = p + q` if `z` is not centered, containing the covariances of parameter estimates.

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 `nPredict` containing the psi weights of the infinite order moving average form of the model.

GetResidual

`public double[] GetResidual()`

Description

Returns the residuals at the final parameter estimate.

This method is only applicable using least-squares algorithm.

Returns

A `double` array of length `z.Length - Math.max(arLags[i]) + length` containing the residuals (including backcasts) at the final parameter estimate point in the first `z.Length - Math.max(arLags[i]) + nb`, where `nb` is the number of values backcast.

SetARLags

`public void SetARLags(int[] arLags)`

Description

The order of the autoregressive parameters.

The elements of `arLags` must be greater than or equal to 1. Default: `arLags` = [1, 2, ..., `p`]

Parameter

`arLags` – An `int` array of length `p` containing the order of the autoregressive parameters.

SetBackcasting

`public void SetBackcasting(int length, double tolerance)`

Description

Sets backcasting option.

Parameters

`length` – An `int` scalar containing the maximum length of backcasting and must be greater than or equal to 0. Default: `length` = 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. Default: `tolerance` = 0.01 * standard deviation of `z`.

SetInitialAREstimates

```
public void SetInitialAREstimates(double[] ar)
```

Description

Sets preliminary autoregressive estimates.

`ar` and `ma` are computed internally if this method is not used. This method is only applicable using least-squares algorithm.

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.

`ar` and `ma` are computed internally if this method is not used. This method is only applicable using least-squares algorithm.

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.

`ar` and `ma` are computed internally if this method is not used. This method is only applicable using least-squares algorithm.

Parameter

`ma` – A `double` array of length `q` containing preliminary estimates of the moving average parameters.

SetMALags

`public void SetMALags(int[] maLags)`

Description

Sets the order of the moving average parameters.

The `maLags` elements must be greater than or equal to 1. Default: `maLags` = [1, 2, ..., `q`]

Parameter

`maLags` – An `int` array of length `q` containing the order of the moving average parameters.

Description

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.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:

Least Squares	Both Method of Moment and Least Squares
	`Center`
`ARLags`	`Method`
`MALags`	`RelativeError`
`Backcasting`	`MaxIterations`
`ConvergenceTolerance`	`MeanEstimate`
`SetInitialEstimates`	`MeanEstimate`
`Residual`	`AutoCovariance`
`SSResidual`	`Variance`
`ParamEstimatesCovariance`	`Constant`
	`AR`
	`MA`

Method of Moments Estimation

Suppose the time series $\{Z_t\}$ is generated by an ARMA $(p,\ q)$ model of the form

$$\phi(B)Z_t = \theta_0 + \theta(B)A_t$$

$$\text{for } \; t \in \{0, \pm 1, \pm 2, \ldots\}$$

Let $\hat{\mu} = \text{MeanEstimate}$ be the estimate of the mean μ of the time series $\{Z_t\}$, where $\hat{\mu}$ equals the following:

$$\hat{\mu} = \begin{cases} \mu & \text{for } \mu \text{ known} \\ \frac{1}{n} \sum\limits_{t=1}^{n} Z_t & \text{for } \mu \text{ unknown} \end{cases}$$

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

$$\hat{\phi} = \left(\hat{\phi}_1, \; \ldots, \; \hat{\phi}_p\right)^T$$

$$\hat{\Sigma}_{ij} = \hat{\sigma}\left(|q + i - j|\right), \;\; i, j = 1, \; \ldots, \; p$$

$$\hat{\sigma}_i = \hat{\sigma}\left(q + i\right), \;\; i = 1, \; \ldots, \; p$$

The overall constant θ_0 is estimated by the following:

$$\hat{\theta}_0 = \begin{cases} \hat{\mu} \;\; \text{for } p = 0 \\ \hat{\mu}\left(1 - \sum\limits_{i=1}^{p} \hat{\phi}_i\right) \;\; \text{for } p > 0 \end{cases}$$

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 ϕ_i for $i = 1, \ldots, p$.

Let $Z'_t = \phi(B)Z_t$. The autocovariances of the derived moving average process $Z'_t = \theta(B)A_t$ are estimated by the following relation:

$$\hat{\sigma}'(k) = \begin{cases} \hat{\sigma}(k) & \text{for } p = 0 \\ \sum_{i=0}^{p}\sum_{j=0}^{p} \hat{\phi}_i\hat{\phi}_j\left(\hat{\sigma}\left(|k+i-j|\right)\right) & \text{for } p \geq 1, \hat{\phi}_0 \equiv -1 \end{cases}$$

The iterative procedure for determining the moving average parameters is based on the relation

$$\sigma(k) = \begin{cases} \left(1 + \theta_1^2 + \ \ldots \ + \theta_q^2\right)\sigma_A^2 & \text{for } k = 0 \\ \left(-\theta_k + \theta_1\theta_{k+1} + \ \ldots \ + \theta_{q-k}\theta_q\right)\sigma_A^2 & \text{for } k \geq 1 \end{cases}$$

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

$$f_j = \sum_{i=0}^{q-j} \tau_i\tau_{i+j} - \hat{\sigma}'(j) \quad \text{for } j = 0, \ 1, \ \ldots, \ q$$

Then, the value of τ at the (i + 1)-th iteration is determined by the following:

$$\tau^{i+1} = \tau^i - \left(T^i\right)^{-1} f^i$$

The estimation procedure begins with the initial value

$$\tau^0 = (\sqrt{\hat{\sigma}'(0)}, \quad 0, \ \ldots, \ 0)^T$$

and terminates at iteration i when either $\left\|f^i\right\|$ is less than `RelativeError` or i equals `MaxIterations`. The moving average parameter estimates are obtained from the final estimate of τ by setting

$$\hat{\theta}_j = -\tau_j / \tau_0 \quad \text{for} \;\; j = 1, \; \ldots, \; q$$

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, μ is the mean of Z_t, and

$$\phi\left(B\right) = 1 - \phi_1 B^{l_\phi(1)} - \phi_2 B^{l_\phi(2)} - \;\ldots\; - \phi_p B^{l_\phi(p)} \qquad \text{for} \;\; p \geq 0$$

$$\theta\left(B\right) = 1 - \theta_1 B^{l_\theta(1)} - \theta_2 B^{l_\theta(2)} - \;\ldots\; - \theta_q B^{l_\theta(q)} \qquad \text{for} \;\; q \geq 0$$

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:

$$l_\phi(i) = i, 1 \leq i \leq p$$

$$l_\theta(j) = j, 1 \le j \le q$$

Consider the sum-of-squares function

$$S_T(\mu, \phi, \theta) = \sum_{-T+1}^{n} [A_t]^2$$

where

$$[A_t] = E\left[A_t \left|(\mu, \phi, \theta, Z)\right.\right]$$

and T is the backward origin. The random shocks $\{A_t\}$ are assumed to be independent and identically distributed

$$N\left(0, \sigma_A^2\right)$$

random variables. Hence, the log-likelihood function is given by

$$l(\mu, \phi, \theta, \sigma_A) = f(\mu, \phi, \theta) - n\ln(\sigma_A) - \frac{S_T(\mu, \phi, \theta)}{2\sigma_A^2}$$

where $f(\mu, \phi, \theta)$ is a function of $\mu, \phi, \text{and}\, \theta$.

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

$$S_\infty(\mu, \phi, \theta) / \left(2\sigma_A^2\right)$$

dominates

$$l\left(\mu, \phi, \theta, \sigma_A^2\right)$$

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, θ_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:

$$1 \le l_\phi(1) \le l_\phi(2) \le \ldots l_\phi(p)$$

$$1 \le l_\theta(1) \le l_\theta(2) \le \ldots \le l_\theta(q)$$

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:

$$l_\phi(i) = i, 1 \le i \le p$$

$$l_\theta(j) = j, 1 \le j \le q$$

The Box-Jenkins forecast at origin t for lead time l of Z_{t+1} is defined in terms of the difference equation

$$\hat{Z}_t\left(l\right) = \theta_0 + \phi_1\left[Z_{t+l-l_{\phi}(1)}\right] + \ \dots\ + \phi_p\left[Z_{t+l-l_{\phi}(p)}\right]$$

$$+\left[A_{t+l}\right] - \theta_1\left[A_{t+l-l_{\theta}(1)}\right]\ -\ ...\ -\ \left[A_{t+l}\right] - \theta_1\left[A_{t+l-l\theta(1)}\right] - ... - \theta_q\left[A_{t+l-l_{\theta}(q)}\right]$$

where the following is true:

$$\left[Z_{t+k}\right] = \begin{cases} Z_{t+k} \ \ \text{for}\ \ k = 0,\ -1,\ -2,\ \dots \\ \hat{Z}_t\left(k\right) \ \ \text{for}\ \ k = 1,\ 2,\ \dots \end{cases}$$

$$\left[A_{t+k}\right] = \begin{cases} Z_{t+k} - \hat{Z}_{t+k-1}\left(1\right) \ \ \text{for}\ \ k = 0,\ -1,\ -2,\ ... \\ 0\ \ \text{for}\ \ k = 1,\ 2,\ ... \end{cases}$$

The $100(1 - \alpha)$ percent probability limits for Z_{t+l} are given by

$$\hat{Z}_t\left(l\right) \pm z_{1/2}\left\{1 + \sum_{j=1}^{l-1}\psi_j^2\right\}^{1/2}\sigma_A$$

where $z_{(1-\alpha/2)}$ is the $100(1 - \alpha/2)$ percentile of the standard normal distribution

$$\sigma_A^2$$

and

$$\left\{\psi_j^2\right\}$$

are the parameters of the random shock form of the difference equation. Note that the forecasts are computed for lead times $l = 1, 2, \dots, L$ at origins $t = (n - b), (n - b + 1), \dots, n$, where L = nPredict and b = BackwardOrigin.

The Box-Jenkins forecasts minimize the mean-square error

$$E\left[Z_{t+l} - \hat{Z}_t\left(l\right)\right]^2$$

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).

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

$$\hat{\theta}_0, \hat{\phi}_1, \hat{\phi}_2, \text{and}\, \hat{\theta}_1$$

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.RelativeError = 0.0;
        arma.MaxIterations = 0;
        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.

```
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, 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.Method = Imsl.Stat.ARMA.ParamEstimation.LeastSquares;
        arma.SetInitialEstimates(arInit, maInit);
        arma.ConvergenceTolerance = 0.125;
        arma.MeanEstimate = 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  -0.733660553488482


   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 "convergence_tolerance" = 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. Method `forecast` in class ARMA computes forecasts and 95-percent probability limits for the forecasts for an ARMA(2, 1) model fit using the method of moments option. With `BackwardOrigin` = 3, `Forecast` method provides forecasts given the data through 1866, 1867, 1868, and 1869, respectively. The deviations from the forecast for computing probability limits, and the psi weights can be used to update forecasts when more data is available. For example, the forecast for the 102-nd observation (year 1871) given the data through the 100-th observation (year 1869) is 77.21; and 95-percent probability limits are given by 77.21 ± 56.30. After observation 101 (Z_{101} for year 1870) is available, the forecast can be updated by using

$$\hat{Z}_t(l) \pm z_{\alpha/2} \left\{ 1 + \sum_{j=1}^{l-1} \psi_j^2 \right\}^{1/2} \sigma_A$$

with the psi weight ($\psi_1 = 1.37$) and the one-step-ahead forecast error for observation $101(Z_{101} - 83.72)$ to give the following:

$$77.21 + 1.37 \times (Z_{101} - 83.72)$$

Since this updated forecast is one step ahead, the 95-percent probability limits are now given by the forecast ± 33.22.

```
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;
using PrintMatrixFormat = Imsl.Math.PrintMatrixFormat;

public 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};
        PrintMatrixFormat pmf = new PrintMatrixFormat();

        ARMA arma = new ARMA(2, 1, z);
        arma.RelativeError = 0.0;
        arma.MaxIterations = 0;
        arma.Compute();

        Console.Out.WriteLine("Method of Moments initial estimates:");
        new PrintMatrix("AR estimates are:  ").Print(arma.GetAR());
        Console.Out.WriteLine();
        new PrintMatrix("MA estimate is:  ").Print(arma.GetMA());
        arma.BackwardOrigin = 3;

        String[] labels = new String[]{"Forecast From 1866",
                                       "Forecast From 1867",
                                       "Forecast From 1868",
                                       "Forecast From 1869"};
        pmf.SetColumnLabels(labels);
        new PrintMatrix("forecasts:  ").Print(pmf, arma.Forecast(12));

        String[] devlabel = new String[]{"Dev. for prob. limits"};
        pmf.SetColumnLabels(devlabel);
        new PrintMatrix().Print(pmf, arma.GetDeviations());

        pmf = new PrintMatrixFormat();
        String[] psilabel = new String[]{"Psi"};
        pmf.SetColumnLabels(psilabel);
        new PrintMatrix().Print(pmf, arma.GetPsiWeights());
    }
}
```

Output

```
Method of Moments initial estimates:
  AR estimates are:
```

	0
0	1.24425777984372
1	-0.575149766040151

MA estimate is:

	0
0	-0.124089747872598

forecasts:

	Forecast From 1866	Forecast From 1867	Forecast From 1868	Forecast From 1869
0	18.2833158907917	16.6150394496618	55.1894123001951	83.7197534904998
1	28.9181987955671	32.0187807491226	62.7607512897864	77.2093688403491
2	41.0100309169882	45.8274629395489	61.8923574323663	63.460943166991
3	49.9387366920847	54.1495649798482	56.4571977708288	50.0988037706777
4	54.0937339010518	56.5623448569975	50.1939146011939	41.380261680911
5	54.12827846595	54.7780099487504	45.5268065977356	38.2173792044093
6	51.781515136941	51.1701275754685	43.3220470047205	39.2964055194313
7	48.8416683089504	47.707251668767	43.2630438046995	42.4581235229259
8	46.5334814013054	45.4736140841138	44.4576955781352	45.77151401381
9	45.3523540994474	44.6860654096231	45.9780860181243	48.0757645397578
10	45.2102804250337	44.9908279786143	47.1827399634897	49.0371504177457
11	45.7128292416607	45.8229896119653	47.8071878011807	48.9080731249673

	Dev. for prob. limits
0	33.2179148279538
1	56.297995631143
2	67.6167546802611
3	70.6432170684592
4	70.7514758474662
5	71.0868521382172
6	71.9073814246285
7	72.5336378185077
8	72.74980142406
9	72.7653184468582
10	72.7779048168612
11	72.8225053997691

	Psi
0	1.36834752771631
1	1.12742729085079
2	0.615805417421561
3	0.117781138936572
4	-0.207630243315585
5	-0.326087340079572
6	-0.286318223936733
7	-0.168704620288894
8	-0.0452361767797933
9	0.0407449580004067
10	0.0767148074728605
11	0.0720185429660699

ARMA.ParamEstimation Enumeration

Summary

Parameter Estimation procedures.

`public enumeration Imsl.Stat.ARMA.ParamEstimation`

Fields

`LeastSquares`
`public Imsl.Stat.ARMA.ParamEstimation LeastSquares`

Description

Indicates autoregressive and moving average parameters are estimated by a least-squares procedure.

`MethodOfMoments`
`public Imsl.Stat.ARMA.ParamEstimation MethodOfMoments`

Description

Indicates autoregressive and moving average parameters are estimated by a method of moments procedure.

Difference Class

Summary

Differences a seasonal or nonseasonal time series.

`public class Imsl.Stat.Difference`

Property

ObservationsLost

`public int ObservationsLost {get; }`

Description

Returns the number of observations lost because of differencing the time series.

Constructor

Difference

`public Difference()`

Description

Constructor for `Difference`.

Methods

Compute

`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

`public void ExcludeFirst(bool exclude)`

Description

Excludes observations lost due to differencing.

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`.

Parameter

`exclude` – A `boolean` specifying whether or not to exclude lost observations due to differencing.

SetOrders

`public void SetOrders(int[] orders)`

Description

Sets the orders for the Difference object.

The elements of `orders` must be greater than or equal to 0.

Parameter

`orders` – An `int` array of length equal to length of `periods`, containing the order of each difference given in periods.

Description

Class `Difference` performs m = `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 $n =$ `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:

$$\Delta_s Z_t = (1 - B^s)\, Z_t = Z_t - Z_{t-s} \quad \text{for } s \geq 0$$

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

$$W_T = \begin{cases} \text{NaN} & \text{for } t = 1, \ldots, n_L \\ \Delta_{s_1}^{d_1} \Delta_{s_2}^{d2} \ldots \Delta_{s_m}^{d_m} Z_t & \text{for } t = n_L + 1, \ldots, n \end{cases}$$

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.

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.

```
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,
                                 158.00, 133.0, 114.0, 140.0};

        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
5
1
-3
-2
10
8
0
0
-8
-4
12
```

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.

```
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
0
-8
-4
12
```

GARCH Class

Summary

Computes estimates of the parameters of a GARCH(p,q) model.

```
public class Imsl.Stat.GARCH
```

Properties

Akaike

`public double Akaike {get; }`

Description

Returns 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.

MaxSigma

`public double MaxSigma {get; set; }`

Description

The value of the upperbound on the first element (sigma) of the array of returned estimated coefficients.

Default = 10.

Sigma

`public double Sigma {get; }`

Description

Returns 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.

`System.ArgumentException` id 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.

`Imsl.Stat.ConstrInconsistentException` id is thrown if the equality constraints are inconsistent

`Imsl.Stat.EqConstrInconsistentException` id is thrown if the equality constraints and the bounds on the variables are found to be inconsistent

`Imsl.Stat.NoVectorXException` id is thrown if no vector X satisfies all of the constraints

`Imsl.Stat.TooManyFunctionEvaluationsException` id is thrown if the number of function evaluations exceeded 1000

`Imsl.Stat.VarsDeterminedException` id is thrown if the variables are determined by the equality constraints

---

GetAR

`public double[] GetAR()`

Description

Returns the estimated values of autoregressive (AR) parameters.

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.

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.

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`.

Returns

A `double` array of size `p + q + 1` containing the estimated values of sigma squared, the AR parameters, and the MA parameters.

Description

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 β_i's and α_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

$$\log(L) = \frac{m}{2} \log(2\pi) - \frac{1}{2} \sum_{t=1}^{m} y_t^2 / \sigma_t^2 - \frac{1}{2} \sum_{t=1}^{m} \log \sigma_t^2,$$

$$\text{where } \sigma_t^2 = \sigma^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2 + \sum_{i=1}^{q} \alpha_i y_{t-i}^2.$$

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

$$2 \times \log(\mathrm{L}) + 2 \times (\mathrm{p} + \mathrm{q} + 1),$$

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. 170), 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).

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.

```
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;
    Imsl.Stat.Random rand = new Imsl.Stat.Random(182198625);
    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;
        sgarch(p, q, m, x, y, wk1, wk2, wk3);

        GARCH garch = new GARCH(p, q, y, xguess);
        garch.Compute();

        Console.Out.WriteLine
            ("Sigma estimate is " + garch.Sigma.ToString("0.000"));
        Console.Out.WriteLine();
        new PrintMatrix("AR estimate is ").Print(garch.GetAR());
        new PrintMatrix("MR estimate is ").Print(garch.GetMA());
        Console.Out.WriteLine("Log-likelihood function value is " +
            garch.LogLikelihood.ToString("0.000"));
        Console.Out.WriteLine("Akaike Information Criterion value is "
             + garch.Akaike.ToString("0.000"));
    }
}
```

Output

```
Sigma estimate is 1.692

   AR estimate is
           0
0  0.244996117092089
1  0.337235176981931

   MR estimate is
           0
0  0.309586601528879

Log-likelihood function value is -2707.073
Akaike Information Criterion value is 5422.146
```

KalmanFilter Class

Summary

Performs Kalman filtering and evaluates the likelihood function for the state-space model.

```
public class Imsl.Stat.KalmanFilter
```

Properties

LogDeterminant

```
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.

In the usual case when P is nonsingular, `LogDeterminant` is the natural log of the determinant of P.

Rank

```
public int Rank {get; }
```

Description

Returns the rank of the variance-covariance matrix for all the observations.

SumOfSquares

```
public double SumOfSquares {get; }
```

Description

Returns the generalized sum of squares.

The estimate of σ^2 is given by `sumOfSquares / n`.

Tolerance

```
public double Tolerance {get; set; }
```

Description

The tolerance used in determining linear dependence.

Default: `tolerance` = 100.0*2.2204460492503131e-16.

Constructor

KalmanFilter

```
public KalmanFilter(double[] b, double[] covb, int rank, double
  sumOfSquaress, double logDeterminant)
```

Description

Constructor for `KalmanFilter`.

`b` is the estimated state vector at time `k` given the observations through time `k`-1.

Parameters

b – A double array containing the estimated state vector.

covb – A double array of size b.Length by b.Length such that covb * σ^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.

sumOfSquaress – 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 * σ^2 is the variance-covariance matrix of the observations.

System.ArgumentException id 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 array 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 dividied 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.

SetQ

`public void SetQ(double[,] q)`

Description

Sets the Q matrix.

Default: There is no error term in the state equation.

Parameter

`q` – A `double` matrix containing the `b.Length` by `b.Length` matrix such that `q` $* \sigma^2$ is the variance-covariance matrix of the error vector in the state equation.

SetTransitionMatrix

`public void SetTransitionMatrix(double[,] t)`

Description

Sets the transition matrix.

Default: `t` = identity matrix

Parameter

`t` – A `double` matrix containing the `b.Length by b.Length` transition matrix in the state equation.

Update

`public void Update(double[] y, double[,] z, double[,] r)`

Description

Performs computation of the update equations.

σ^2 is a positive unknown scalar. Only elements in the upper triangle of `r` are referenced.

Parameters

`y` – A `double` array containing the observations.

`z` – A `double` matrix containing the `y.Length by b.Length` matrix relating the observations to the state vector in the observation equation.

`r` – A `double` matrix containing the `y.Length by y.Length` matrix such that `r` * σ^2 is the variance-covariance matrix of errors in the observation equation.

Description

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, σ^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 `covb`, respectively. During the k-th invocation, `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 `covv`. The "start-up values" needed on the first invocation of `KalmanFilter` are

$$\hat{\beta}_{1|0} = \mu_1$$

and $C_{1|0} = Q_1$ input via `b` and `covb`, respectively. Computations for the k-th invocation are completed by `KalmanFilter` computing the one-step-ahead estimate

$$\hat{\beta}_{k+1|k}$$

along with $C_{k+1|k}$ given by the *prediction equations*:

$$\hat{\beta}_{k+1|k} = T_{k+1}\hat{\beta}_{k|k}$$

$$C_{k+1|k} = T_{k+1}C_{k|k}T_{k+1}^T + Q_{k+1}$$

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, σ^2. The maximum likelihood estimate of σ^2 based on the observations $y_1, y_2, \ldots, y_m$, is given by

$$\hat{\sigma}^2 = SS/N$$

where

$$N = \sum\nolimits_{k=1}^{m} n_k \text{ and } SS = \sum\nolimits_{k=1}^{m} v_k^T H_k^{-1} v_k$$

N and SS are the input/output arguments `n` and `sumOfSquares`.

If σ^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 θ. In this case, `KalmanFilter` can be used in conjunction with an optimization class (see class `MinUnconMultiVar`, IMSL C# Library Math namespace), to obtain a maximum likelihood estimate of θ. The natural logarithm of the likelihood function for $y_1, y_2, \ldots, y_m$ differs by no more than an additive constant from

$$L(\theta, \sigma^2; y_1, y_2,\, \ldots,\, y_m) = -\frac{1}{2} N \ln \sigma^2 - \frac{1}{2}\sum_{k=1}^{m} \ln[\det(H_k)] - \frac{1}{2}\sigma^{-2}\sum_{k=1}^{m} v_k^T H_k^{-1} v_k$$

(Harvey 1981, page 14, equation 2.21).

Here,

$$\sum_{k=1}^{m} \ln[\det(H_k)]$$

(stored in 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 θ and σ^2 produces maximum likelihood estimates. Equivalently, minimization of $-2L_c(\theta; y_1, y_2, \ldots, y_m)$ where

$$L_c(\theta; y_1, y_2, \ \ldots, \ y_m) = -\frac{1}{2} N \ln\left(\frac{SS}{N}\right) - \frac{1}{2}\sum_{k=1}^{m} \ln[\det(H_k)]$$

produces maximum likelihood estimates

$$\hat{\theta} \text{ and } \hat{\sigma}^2 = SS/N$$

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

$$\hat{\sigma}^2(\theta) = SS(\theta)/N$$

minimizes $-2L(\theta, \sigma^2; y_1, y_2, \ldots, y_m)$ for all θ, consequently,

$$\hat{\sigma}^2 (\theta)$$

can be substituted for σ^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 $\sum_{k=1}^{m} \text{rank}\,(H_k)$

Maximum likelihood estimation of parameters in the Kalman filter is discussed by Sallas and Harville (1988) and Harvey (1981, pages 111-113).

Example: Kilman 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_ks are identically and independently distributed normal with mean 0 and variance σ^2, the w_ks 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}\t{2:0.000}\t{3:0.000}\t{4}\t{5:0.000}" +
        "\t{6:0.000}\t{7:0.000}\t{8:0.000}";

    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[]{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));

        for (int i = 0; i < nobs; i++)
        {
            double[] y = new double[]{ydata[i]};
            KalmanFilter kalman =
                new KalmanFilter(b, covb, rank, ss, logDeterminant);
            kalman.Update(y, z, r);
            kalman.Filter();
            b = kalman.GetStateVector();
            covb = kalman.GetCovB();
            rank = kalman.Rank;
            ss = kalman.SumOfSquares;
            logDeterminant = kalman.LogDeterminant;
            double[] v = kalman.GetPredictionError();
            double[,] covv = kalman.GetCovV();
            argFormat[0] = i;
            argFormat[1] = i;
            argFormat[2] = b[0];
            argFormat[3] = covb[0];
            argFormat[4] = rank;
            argFormat[5] = ss;
            argFormat[6] = logDeterminant;
            argFormat[7] = v[0];
            argFormat[8] = covv[0,0];
            Console.Out.WriteLine(String.Format(format, argFormat));

            kalman =
                new KalmanFilter(b, covb, rank, ss, logDeterminant);
            kalman.SetTransitionMatrix(t);
            kalman.SetQ(q);
            kalman.Filter();
            b = kalman.GetStateVector();
            covb = kalman.GetCovB();
            rank = kalman.Rank;
            ss = kalman.SumOfSquares;
            logDeterminant = kalman.LogDeterminant;
            argFormat[0] = i + 1;
            argFormat[1] = i;
            argFormat[2] = b[0];
            argFormat[3] = covb[0];
            argFormat[4] = rank;
            argFormat[5] = ss;
            argFormat[6] = logDeterminant;
            argFormat[7] = v[0];
            argFormat[8] = covv[0,0];
            Console.Out.WriteLine(String.Format(format, argFormat));
        }
    }
}
```

Output

```
k/j b cov(b) rank ss ln(det) v cov(v)
0/0 4.376 0.941 1 0.009 2.833 0.400 17.000
1/0 4.376 4.941 1 0.009 2.833 0.400 17.000
1/1 4.063 0.832 2 0.033 4.615 -0.376 5.941
2/1 4.063 4.832 2 0.033 4.615 -0.376 5.941
2/2 3.597 0.829 3 0.088 6.378 -0.563 5.832
3/2 3.597 4.829 3 0.088 6.378 -0.563 5.832
3/3 4.428 0.828 4 0.260 8.141 1.003 5.829
4/3 4.428 4.828 4 0.260 8.141 1.003 5.829
```

Chapter 19: Multivariate Analysis

Types

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, μ is the p vector of variable means, Λ 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

Summary

Perform a K-means (centroid) cluster analysis.

`public class Imsl.Stat.ClusterKMeans`

Property

MaxIterations

`public int MaxIterations {get; set; }`

Description

The maximum number of iterations.

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.

`System.ArgumentException` id 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.

`Imsl.Stat.NoConvergenceException` id is thrown if convergence did not occur within the maximum number of iterations

`Imsl.Stat.ClusterNoPointsException` id 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.

Returns

An `int` array containing the number of observations in each cluster.

GetClusterMembership

`public int[] GetClusterMembership()`

Description

Returns the cluster membership for each observation.

Cluster membership 1 indicates the observation belongs to cluster 1, cluster membership 2 indicates the observation belongs to cluster 2, etc.

Returns

An `int` array containing the cluster membership for each observation.

GetClusterSSQ

`public double[] GetClusterSSQ()`

Description

Returns the within sum of squares for each cluster.

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.

Default: `Frequencies[]` $= 1$.

Parameter

`frequencies` – A `double` array of size `x.GetLength(0)` containing the frequency for each observation.

SetWeights

`public void SetWeights(double[] weights)`

Description

Sets the weight for each observation.

Default: `Weights[]` = 1.

Parameter

`weights` – A `double` array of size `x.GetLength(0)` containing the weight for each observation.

Description

`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, *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,

$$\phi = \sum_{i=1}^{K} \sum_{j=1}^{p} \sum_{m=1}^{n_i} f_{\nu_{im}} w_{\nu_{im}} \delta_{\nu_{im},j} \left(x_{\nu_{im},j} - \bar{x}_{ij} \right)^2$$

where ν_{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 ν_{im} is missing, otherwise δ 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.

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.

```
/*
* -------------------------------------------------------------------------
*   Copyright (c) 1999 Visual Numerics Inc. All Rights Reserved.
*
*   This software is confidential information which is proprietary to
*   and a trade secret of Visual Numerics, Inc.  Use, duplication or
*   disclosure is subject to the terms of an appropriate license
*   agreement.
*
*   VISUAL NUMERICS MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE
*   SUITABILITY OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING
*   BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY,
*   FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. VISUAL
*   NUMERICS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE
*   AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR
*   ITS DERIVATIVES.
*--------------------------------------------------------------------------
*/
using System;
using Imsl.Stat;
using Imsl.Math;

public 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},
                          {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},
                          {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},
{6.300, 3.400, 5.600, 2.400},
{6.400, 3.100, 5.500, 1.800},
{6.000, 3.000, 4.800, 1.800},
{6.900, 3.100, 5.400, 2.100},
{6.700, 3.100, 5.600, 2.400},
{6.900, 3.100, 5.100, 2.300},
```

```
                        {5.800, 2.700, 5.100, 1.900},
                        {6.800, 3.200, 5.900, 2.300},
                        {6.700, 3.300, 5.700, 2.500},
                        {6.700, 3.000, 5.200, 2.300},
                        {6.300, 2.500, 5.000, 1.900},
                        {6.500, 3.000, 5.200, 2.000},
                        {6.200, 3.400, 5.400, 2.300},
                        {5.900, 3.000, 5.100, 1.800}};


        double[,] cs = {{5.100, 3.500, 1.400, 0.200},
                         {7.000, 3.200, 4.700, 1.400},
                         {6.300, 3.300, 6.000, 2.500}};


        ClusterKMeans kmean = new ClusterKMeans(x, cs);

        double[,] cm = kmean.Compute();
        double[] wss = kmean.GetClusterSSQ();
        int[] ic = kmean.GetClusterMembership();
        int[] nc = kmean.GetClusterCounts();

        PrintMatrix pm = new PrintMatrix("Cluster Means");

        PrintMatrixFormat pmf = new PrintMatrixFormat();
        pmf.NumberFormat = "0.0000";
        pm.Print(pmf, cm);

        new PrintMatrix("Cluster Membership").Print(ic);
        new PrintMatrix("Sum of Squares").Print(wss);
        new PrintMatrix("Number of observations").Print(nc);
    }
}
```

Output

```
           Cluster Means
     0       1       2       3
0  5.0060  3.4280  1.4620  0.2460
1  5.9016  2.7484  4.3935  1.4339
2  6.8500  3.0737  5.7421  2.0711

Cluster Membership
     0
  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
 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
 61   2
 62   2
 63   2
 64   2
```

```
 65   2
 66   2
 67   2
 68   2
 69   2
 70   2
 71   2
 72   2
 73   2
 74   2
 75   2
 76   2
 77   3
 78   2
 79   2
 80   2
 81   2
 82   2
 83   2
 84   2
 85   2
 86   2
 87   2
 88   2
 89   2
 90   2
 91   2
 92   2
 93   2
 94   2
 95   2
 96   2
 97   2
 98   2
 99   2
100   3
101   2
102   3
103   3
104   3
105   3
106   2
107   3
108   3
109   3
110   3
111   3
112   3
113   2
114   2
115   3
116   3
117   3
118   3
119   2
120   3
```

```
121  2
122  3
123  2
124  3
125  3
126  2
127  2
128  3
129  3
130  3
131  3
132  3
133  2
134  3
135  3
136  3
137  3
138  2
139  3
140  3
141  3
142  2
143  3
144  3
145  3
146  2
147  3
148  3
149  2

   Sum of Squares
          0
0  15.151
1  39.8209677419355
2  23.8794736842105

Number of observations
   0
0  50
1  62
2  38
```

Dissimilarities Class

Summary

Computes a matrix of dissimilarities (or similarities) between the columns (or rows) of a matrix.

```
public class Imsl.Stat.Dissimilarities
```

Property

DistanceMatrix

```
virtual public double[,] DistanceMatrix {get; }
```

Description

The distance matrix.

Constructors

Dissimilarities

```
public Dissimilarities(double[,] x, int distanceMethod, int distanceScale,
  int iRow)
```

Description

Constructor for `Dissimilarities`.

Acceptable values of *distanceMethod* are 1, 2, ..., 8. See *Remarks* section of the `Dissimilarities` documentation for a description of these methods.

distanceScale	**Method**
0	No scaling is performed.
1	Scale each column (row if *iRow=1*) by the standard deviation of the column(row).
2	Scale each column (row if *iRow=1*) by the range of the column (row).

If $iRow = 1$, distances are computed between the rows of x. Otherwise, distances between the columns of x are computed.

Parameters

`x` – A `double` matrix containing the data input matrix.

`distanceMethod` – An `int` identifying the method to use in computing the dissimilarities or similarities.

`distanceScale` – An `int` containing the scaling option.

`iRow` – An `int` identifying whether distances are computed between rows or columns of x.

`Imsl.Stat.ScaleFactorZeroException` id is thrown when computations cannot continue because a scale factor is zero.

`Imsl.Stat.NoPositiveVarianceException` id is thrown when no variable has positive variance

`Imsl.Stat.ZeroNormException` id is thrown when the Euclidean norm of a column is equal to zero

Dissimilarities

```
public Dissimilarities(double[,] x, int distanceMethod, int distanceScale,
  int iRow, int[] indexArray)
```

Description

Constructor for `Dissimilarities`.

Acceptable values of *distanceMethod* are 1, 2, ..., 8. See *Remarks* section of the `Dissimilarities` documentation for a description of these methods.

distanceScale	**Method**
0	No scaling is performed.
1	Scale each column (row if *iRow=1*) by the standard deviation of the column(row).
2	Scale each column (row if *iRow=1*) by the range of the column (row).

If $iRow = 1$, distances are computed between the rows of x. Otherwise, distances between the columns of x are computed.

Parameters

`x` – A `double` matrix containing the data input matrix.

`distanceMethod` – An `int` identifying the method to use in computing the dissimilarities or similarities.

`distanceScale` – An `int` containing the scaling option.

`iRow` – An `int` identifying whether distances are computed between rows or columns of x.

`indexArray` – An `int` array containing the indices of the rows (columns if *iRow* is 1) to use in computing the distance measure.

`Imsl.Stat.ScaleFactorZeroException` id is thrown when computations cannot continue because a scale factor is zero.

`Imsl.Stat.NoPositiveVarianceException` id is thrown when no variable has positive variance

`Imsl.Stat.ZeroNormException` id is thrown when the Euclidean norm of a column is equal to zero

Description

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, set *iRow* to 1 when calling the `Dissimilarities` constructor.

For *distanceMethod* = 0 to 2, each row of x is first scaled according to the value of *distanceScale*. 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 *distanceScale* is 0, no scaling is performed, and the parameters in the following discussion are all 1.0. Once the scaling value (if any) has been computed, the distance between column i and column j is computed via the difference vector $z_k = \frac{(x_k - y_k)}{s_k}, i = 1, \ldots, ndstm$, 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 metrics 0 to 2 are defined as:

`distanceMethod`	**Metric**
0	Euclidean distance (L_2 norm)
1	Sum of the absolute differences (L_1 norm)
2	Maximum difference (L_∞ norm)

Distance measures corresponding to `distanceMethod` = 3 to 8 do not allow for scaling.

`distanceMethod`	**Metric**
3	Mahalanobis distance
4	Absolute value of the cosine of the angle between the vectors
5	Angle in radians (0, pi) between the lines through the origin defined by the vectors
6	Correlation coefficient
7	Absolute value of the correlation coefficient
8	Number of exact matches, where $x_i = y_i$.

For the Mahalanobis distance, any variable used in computing the distance measure that is (numerically) linearly dependent upon the previous variables in the *indexArray* vector is omitted from the distance measure.

Example: Dissimilarities

The following example illustrates the use of Dissimilarities for computing the Euclidean distance between the rows of a matrix:

```
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}};
        int distanceMethod = 0;
        int distanceScale = 0;
        int iRow = 1;
        Dissimilarities dist = new Dissimilarities(x, distanceMethod, distanceScale, iRow);
                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
```

ClusterHierarchical Class

Summary

Performs a hierarchical cluster analysis from a distance matrix.

`public class Imsl.Stat.ClusterHierarchical`

Properties

ClusterLeftSons

`virtual public int[] ClusterLeftSons {get; }`

Description

The left sons of each merged cluster.

ClusterLevel

`virtual public double[] ClusterLevel {get; }`

Description

The level at which the clusters are joined.

Element [*k*-1] contains the distance (or similarity) level at which cluster $npt + k$ was formed. If the original data in *dist* was transformed, the inverse transformation is applied to the returned values.

ClusterRightSons

```
virtual public int[] ClusterRightSons {get; }
```

Description

The right sons of each merged cluster.

Constructor

ClusterHierarchical

```
public ClusterHierarchical(double[,] dist, int method, int transform)
```

Description

Constructor for `ClusterHierarchical`.

On input, only the upper triangular part of *dist* needs to be present. `ClusterHierarchical` saves the upper triangular part in the lower triangle. On return, the upper triangular part of *dist* is restored, and the matrix is made symmetric.

method	**Description**
0	Single linkage (minimum distance).
1	Complete linkage (maximum distance).
2	Average distance within (average distance between objects within he merged cluster).
3	Average distance between (average distance between objects in the two clusters).
4	Ward's method (minimize the within-cluster sums of squares). For Ward's method, the elements of *dist* are assumed to be Euclidean distances.

transform	**Description**
0	No transformation is required. The elements of*dist* are distances.
1	Convert similarities to distances by multiplying -1.0.
2	Convert similarities (usually correlations) to distances by taking the reciprocal of the absolute value.

Parameters

`dist` – A `double` symmetric matrix containing the distance (or similarity) matrix.

`method` – An `int` identifying the clustering method to use.

`transform` – An `int` identifying the type of transformation applied to the measures in *dist*.

Methods

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.

Description

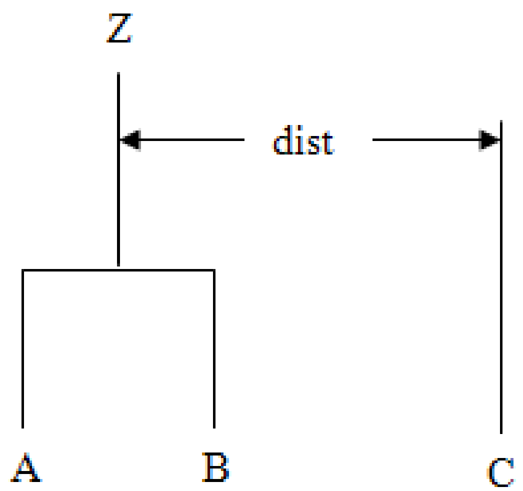
Class `ClusterHierarchical` conducts a hierarchical cluster analysis based upon a distance matrix, or by appropriate use of the argument *transform*, based upon a similarity matrix. Only the upper triangular part of the *dist* matrix is required as input.

Hierarchical clustering in `ClusterHierarchical` proceeds as follows:

1. Initially, each data point is considered to be a cluster, numbered 1 to n = *npt*, where *npt* is the number of rows in *dist*.
2. If the data matrix contains similarities, they are converted to distances by the method specified by the argument *transform*. Set $k = 1$.
3. 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* .
4. 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.

5. 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 argument *method* 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.



`method`	**Description**
0	Single linkage (minimum distance). The distance from Z to C is the minimum of the distances (A to C, B to C).
1	Complete linkage (maximum distance). The distance from Z to C is the maximum of the distances (A to C, B to C).
2	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).
3	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).
4	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).

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 *npt*, where *npt* is the number of rows in *dist*. The *npt* - 1 clusters formed by merging clusters are numbered *npt* + 1 to *npt* + (*npt* - 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 argument *transform*, with *transform* = 2.
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. 552) may be used to compute the matrix *dist* for either the variables or observations.

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.

```
using System;
using Imsl.Math;
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},
{ 5.1, 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},
{ 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, 2.5},
                { 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, 0, 1, 1);
        ClusterHierarchical clink = new ClusterHierarchical(dist.DistanceMatrix, 2, 0);

        int nClusters = 5;
        int[] iclus = clink.GetClusterMembership(nClusters);
        int[] nclus = clink.GetObsPerCluster(nClusters);
        System.Console.Out.WriteLine("Cluster Membership");
        for (int i = 0; i < 15; i++)
        {
            for (int j = 0; j < 10; j++)
                Console.Out.Write(iclus[i * 10 + j] + " ");
            Console.Out.WriteLine();
        }

        System.Console.Out.WriteLine("Observations Per Cluster");
        for (int i = 0; i < nClusters; i++)
            System.Console.Out.Write(nclus[i] + " ");
        System.Console.Out.WriteLine();
    }
}
```

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
5 5 5 5 5 5 5 5 5 5
3 3 3 4 3 4 3 4 3 4
4 3 4 3 4 3 4 4 4 4
3 3 3 3 3 3 3 3 3 4
4 4 4 3 4 3 3 4 4 4
4 3 4 4 4 4 4 3 4 4
2 3 2 3 2 1 4 1 3 2
2 3 2 3 3 2 3 2 1 4
2 3 1 3 2 1 3 3 3 1
1 2 3 3 3 1 2 3 3 2
2 2 3 2 2 2 3 3 2 3
Observations Per Cluster
8 19 44 29 50
```

FactorAnalysis Class

Summary

Performs Principal Component Analysis or Factor Analysis on a covariance or correlation matrix.

`public class Imsl.Stat.FactorAnalysis`

Properties

ConvergenceCriterion1

`public double ConvergenceCriterion1 {get; set; }`

Description

The convergence criterion used to terminate the iterations.

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.

ConvergenceCriterion2

```
public double ConvergenceCriterion2 {get; set; }
```

Description

The convergence criterion used to switch to exact second derivatives.

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.

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.

For the principal component and principal factor methods, the factor loading estimates are computed as

$$\hat{\Gamma}\hat{\Delta}^{-1/2}$$

where Γ and the diagonal matrix Δ 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 Ψ 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 `Variances` 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 `Variances` 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 Ψ and Γ. (In the principal factor method, Ψ is assumed to be known. Given Ψ, estimates for Λ 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 Σ 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 Σ. This is written as $\Phi_u l = .5trace((S - \Sigma)^2)$.

Generalized least-squares and maximum likelihood estimates are asymptotically equivalent methods. Maximum likelihood estimates maximize the (normal theory) likelihood $\{\Phi_m l = trace(\Sigma^{-1}S) - log(|\Sigma^{-1}S|)\}$. while generalized least squares optimizes the function $\Phi_g s = trace(\Sigma S^{-1} - I)^2$.

In all three methods, a two-stage optimization procedure is used. This proceeds by first solving the likelihood equations for Λ in terms of Ψ and substituting the solution into the likelihood. This gives a criterion $\Phi(\Psi, \Lambda(\Psi))$, which is optimized with respect to Ψ. In the second stage, the estimates

$$\hat{\Lambda}$$

are obtained from the estimates for Ψ.

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 `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 `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 `stat[3]`.

The Tucker and Lewis (1973) reliability coefficient, ρ, is returned in `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 `cov`, p is the number of variables, k is the number of factors, Φ 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 $D^2 = (diag(S^{-1}))^{-1}$ is computed where the operator "diag" results in a matrix consisting of the diagonal elements of its argument, and S is the sample covariance (correlation) matrix. Then, the eigenvalues Λ and eigenvectors Γ 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 ξ 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 ξ. In order to obtain these estimates, the iterative algorithm of Kaiser and Caffrey (1965) is used.

MaxIterations

```
public int MaxIterations {get; set; }
```

Description

The maximum number of iterations in the iterative procedure.

By default, `MaxIterations` is set to 60. `MaxIterations` is not referenced for factor loading methods PrincipalComponent, PrincipalFactor, or ImageFactorAnalysis.

MaxStep

```
public int MaxStep {get; set; }
```

Description

The maximum number of step halvings allowed during an iteration.

If this property is not set, `MaxStep` is set to 8. `MaxStep` is not referenced for `PrincipalComponent`, `PrincipalFactor`, `ImageFactorAnalysis`, or `AlphaFactorAnalysis` methods.

VarianceEstimationMethod

```
public int VarianceEstimationMethod {get; set; }
```

Description

The variance estimation method.

By default, `VarianceEstimationMethod` is set to 1.

init	Method
0	Initial estimates are taken as the constant 1-nfactors/(2*nvar) divided by the diagonal elements of the inverse of input matrix `cov`.
1	Initial estimates are input by the user in vector `uniq`.

Note that when the factor loading estimation method is PrincipalComponent, the initial estimates in `uniq` are reset to 0.0.

Constructor

FactorAnalysis

```
public FactorAnalysis(double[,] cov, Imsl.Stat.FactorAnalysis.MatrixType
  matrixType, int nfactors)
```

Description

Constructor for `FactorAnalysis`.

`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.

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.

`System.ArgumentException` id is thrown if `x.GetLength(0)`, and `x.GetLength(1)` are equal to 0

Methods

GetCorrelations

`public double[,] GetCorrelations()`

Description

Returns the correlations of the principal components.

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.

Returns

A `double` matrix containing the correlations of the principal components with the observed/standardized variables.

`Imsl.Stat.RankException` id is thrown if the rank of the covariance matrix is less than the number of factors.

`Imsl.Stat.NoDegreesOfFreedomException` id is thrown if there are no degrees of freedom for the significance testing.

`Imsl.Stat.NotSemiDefiniteException` id is thrown if the Hessian matrix not semi-definite.

`Imsl.Stat.NotPositiveSemiDefiniteException` id is thrown if the covariance matrix is not positive semi-definite.

`Imsl.Stat.NotPositiveDefiniteException` id is thrown if the covariance matrix is not positive definite because a variable is linearly releated to other variables.

`Imsl.Stat.SingularException` id is thrown if the covariance matrix is singular.

`Imsl.Stat.BadVarianceException` id is thrown if the input variance is not in the allowed range.

`Imsl.Stat.EigenvalueException` id is thrown if an error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

`Imsl.Stat.NonPositiveEigenvalueException` id 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.

`Imsl.Stat.RankException` id is thrown if the rank of the covariance matrix is less than the number of factors.

`Imsl.Stat.NoDegreesOfFreedomException` id is thrown if there are no degrees of freedom for the significance testing.

`Imsl.Stat.NotSemiDefiniteException` id is thrown if the Hessian matrix not semi-definite.

`Imsl.Stat.NotPositiveSemiDefiniteException` id is thrown if the covariance matrix is not positive semi-definite.

`Imsl.Stat.NotPositiveDefiniteException` id is thrown if the covariance matrix is not positive definite because a variable is linearly releated to other variables.

`Imsl.Stat.SingularException` id is thrown if the covariance matrix is singular.

`Imsl.Stat.BadVarianceException` id is thrown if the input variance is not in the allowed range.

`Imsl.Stat.EigenvalueException` id is thrown if an error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

`Imsl.Stat.NonPositiveEigenvalueException` id 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.

The parameter updates are only meaningful for the common factor model. The parameter updates are set to 0.0 for the principal component model.

Returns

A `double` array containing the parameter updates when convergence was reached (or the iterations terminated).

`Imsl.Stat.RankException` id is thrown if the rank of the covariance matrix is less than the number of factors.

`Imsl.Stat.NoDegreesOfFreedomException` id is thrown if there are no degrees of freedom for the significance testing.

`Imsl.Stat.NotSemiDefiniteException` id is thrown if the Hessian matrix not semi-definite.

`Imsl.Stat.NotPositiveSemiDefiniteException` id is thrown if the covariance matrix is not positive semi-definite.

`Imsl.Stat.NotPositiveDefiniteException` id is thrown if the covariance matrix is not positive definite because a variable is linearly releated to other variables.

`Imsl.Stat.SingularException` id is thrown if the covariance matrix is singular.

`Imsl.Stat.BadVarianceException` id is thrown if the input variance is not in the allowed range.

`Imsl.Stat.EigenvalueException` id is thrown if an error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

`Imsl.Stat.NonPositiveEigenvalueException` id 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.

Valid for the principal component model.

Returns

A `double` array containing the total variance explained by each principal component.

`Imsl.Stat.RankException` id is thrown if the rank of the covariance matrix is less than the number of factors.

`Imsl.Stat.NoDegreesOfFreedomException` id is thrown if there are no degrees of freedom for the significance testing.

`Imsl.Stat.NotSemiDefiniteException` id is thrown if the Hessian matrix not semi-definite.

`Imsl.Stat.NotPositiveSemiDefiniteException` id is thrown if the covariance matrix is not positive semi-definite.

`Imsl.Stat.NotPositiveDefiniteException` id is thrown if the covariance matrix is not positive definite because a variable is linearly releated to other variables.

`Imsl.Stat.SingularException` id is thrown if the covariance matrix is singular.

`Imsl.Stat.BadVarianceException` id is thrown if the input variance is not in the allowed range.

`Imsl.Stat.EigenvalueException` id is thrown if an error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

`Imsl.Stat.NonPositiveEigenvalueException` id 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

`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.

`Imsl.Stat.RankException` id is thrown if the rank of the covariance matrix is less than the number of factors.

`Imsl.Stat.NoDegreesOfFreedomException` id is thrown if there are no degrees of freedom for the significance testing.

`Imsl.Stat.NotSemiDefiniteException` id is thrown if the Hessian matrix not semi-definite.

`Imsl.Stat.NotPositiveSemiDefiniteException` id is thrown if the covariance matrix is not positive semi-definite.

`Imsl.Stat.NotPositiveDefiniteException` id is thrown if the covariance matrix is not positive definite because a variable is linearly releated to other variables.

`Imsl.Stat.SingularException` id is thrown if the covariance matrix is singular.

`Imsl.Stat.BadVarianceException` id is thrown if the input variance is not in the allowed range.

`Imsl.Stat.EigenvalueException` id is thrown if an error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

`Imsl.Stat.NonPositiveEigenvalueException` id 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

`public double[] GetStatistics()`

Description

Returns statistics.

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.

i	Statistics[i]
0	Value of the function minimum.
1	Tucker reliability coefficient.
2	Chi-squared test statistic for testing that the number of factors in the model are adequate for the data.
3	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.
4	Probability of a greater chi-squared statistic.
5	Number of iterations.

Returns

A `double` array containing output statistics.

`Imsl.Stat.RankException` id is thrown if the rank of the covariance matrix is less than the number of factors.

`Imsl.Stat.NoDegreesOfFreedomException` id is thrown if there are no degrees of freedom for the significance testing.

`Imsl.Stat.NotSemiDefiniteException` id is thrown if the Hessian matrix not semi-definite.

`Imsl.Stat.NotPositiveSemiDefiniteException` id is thrown if the covariance matrix is not positive semi-definite.

`Imsl.Stat.NotPositiveDefiniteException` id is thrown if the covariance matrix is not positive definite because a variable is linearly releated to other variables.

`Imsl.Stat.SingularException` id is thrown if the covariance matrix is singular.

`Imsl.Stat.BadVarianceException` id is thrown if the input variance is not in the allowed range.

`Imsl.Stat.EigenvalueException` id is thrown if an error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

`Imsl.Stat.NonPositiveEigenvalueException` id 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.

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.

Returns

A `double` array containing the eigenvalues of the matrix from which the factors were extracted ordered from largest to smallest.

`Imsl.Stat.RankException` id is thrown if the rank of the covariance matrix is less than the number of factors.

`Imsl.Stat.NoDegreesOfFreedomException` id is thrown if there are no degrees of freedom for the significance testing.

`Imsl.Stat.NotSemiDefiniteException` id is thrown if the Hessian matrix not semi-definite.

`Imsl.Stat.NotPositiveSemiDefiniteException` id is thrown if the covariance matrix is not positive semi-definite.

`Imsl.Stat.NotPositiveDefiniteException` id is thrown if the covariance matrix is not positive definite because a variable is linearly releated to other variables.

`Imsl.Stat.SingularException` id is thrown if the covariance matrix is singular.

`Imsl.Stat.BadVarianceException` id is thrown if the input variance is not in the allowed range.

`Imsl.Stat.EigenvalueException` id is thrown if an error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

`Imsl.Stat.NonPositiveEigenvalueException` id 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

`public double[] GetVariances()`

Description

Returns the unique variances.

Returns

A `double` array containing the unique variances.

`Imsl.Stat.RankException` id is thrown if the rank of the covariance matrix is less than the number of factors.

`Imsl.Stat.NoDegreesOfFreedomException` id is thrown if there are no degrees of freedom for the significance testing.

`Imsl.Stat.NotSemiDefiniteException` id is thrown if the Hessian matrix not semi-definite.

`Imsl.Stat.NotPositiveSemiDefiniteException` id is thrown if the covariance matrix is not positive semi-definite.

`Imsl.Stat.NotPositiveDefiniteException` id is thrown if the covariance matrix is not positive definite because a variable is linearly releated to other variables.

`Imsl.Stat.SingularException` id is thrown if the covariance matrix is singular.

`Imsl.Stat.BadVarianceException` id is thrown if the input variance is not in the allowed range.

`Imsl.Stat.EigenvalueException` id is thrown if an error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

`Imsl.Stat.NonPositiveEigenvalueException` id 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.

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.

Returns

A `double` matrix containing the eigenvectors of the matrix from which the factors were extracted.

`Imsl.Stat.RankException` id is thrown if the rank of the covariance matrix is less than the number of factors.

`Imsl.Stat.NoDegreesOfFreedomException` id is thrown if there are no degrees of freedom for the significance testing.

`Imsl.Stat.NotSemiDefiniteException` id is thrown if the Hessian matrix not semi-definite.

`Imsl.Stat.NotPositiveSemiDefiniteException` id is thrown if the covariance matrix is not positive semi-definite.

`Imsl.Stat.NotPositiveDefiniteException` id is thrown if the covariance matrix is not positive definite because a variable is linearly releated to other variables.

`Imsl.Stat.SingularException` id is thrown if the covariance matrix is singular.

`Imsl.Stat.BadVarianceException` id is thrown if the input variance is not in the allowed range.

`Imsl.Stat.EigenvalueException` id is thrown if an error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

`Imsl.Stat.NonPositiveEigenvalueException` id 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.

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.

Parameter

`uniq` – A `double` array of length nvar containing the unique variances.

Description

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 Γ 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 Σ is the $p \times p$ population covariance matrix. Λ is the $p \times k$ matrix of factor loadings relating the factors f to the observed variables x, and Ψ 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 Ψ 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.

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.

```
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
     0
0  4.6769
1  1.2640
2  0.8444
3  0.5550
4  0.4471
5  0.4291
6  0.3102
7  0.2770
8  0.1962

 Percents
     0
```

```
0  0.5197
1  0.6601
2  0.7539
3  0.8156
4  0.8653
5  0.9130
6  0.9474
7  0.9782
8  1.0000

Standard Errors
     0
0  0.6498
1  0.1771
2  0.0986
3  0.0879
4  0.0882
5  0.0890
6  0.0944
7  0.0994
8  0.1113

                                  Eigenvectors
     0        1        2        3        4        5        6        7        8
0  0.3462  -0.2354   0.1386  -0.3317  -0.1088   0.7974   0.1735  -0.1240  -0.0488
1  0.3526  -0.1108  -0.2795  -0.2161   0.7664  -0.2002   0.1386  -0.3032  -0.0079
2  0.2754  -0.2697  -0.5585   0.6939  -0.1531   0.1511   0.0099  -0.0406  -0.0997
3  0.3664   0.4031   0.0406   0.1196   0.0017   0.1152  -0.4022  -0.1178   0.7060
4  0.3144   0.5022  -0.0733  -0.0207  -0.2804  -0.1796   0.7295   0.0075   0.0046
5  0.3455   0.4553   0.1825   0.1114   0.1202   0.0696  -0.3742   0.0925  -0.6780
6  0.3487  -0.2714  -0.0725  -0.3545  -0.5242  -0.4355  -0.2854  -0.3408  -0.1089
7  0.2407  -0.3159   0.7383   0.4329   0.0861  -0.1969   0.1862  -0.1623   0.0505
8  0.3847  -0.2533  -0.0078  -0.1468   0.0459  -0.1498  -0.0251   0.8521   0.1225

                           Unrotated Factor Loadings
     0        1        2        3        4        5        6        7        8
0  0.7487  -0.2646   0.1274  -0.2471  -0.0728   0.5224   0.0966  -0.0652  -0.0216
1  0.7625  -0.1245  -0.2568  -0.1610   0.5124  -0.1312   0.0772  -0.1596  -0.0035
2  0.5956  -0.3032  -0.5133   0.5170  -0.1024   0.0990   0.0055  -0.0214  -0.0442
3  0.7923   0.4532   0.0373   0.0891   0.0012   0.0755  -0.2240  -0.0620   0.3127
4  0.6799   0.5646  -0.0674  -0.0154  -0.1875  -0.1177   0.4063   0.0039   0.0021
5  0.7472   0.5119   0.1677   0.0830   0.0804   0.0456  -0.2084   0.0487  -0.3003
6  0.7542  -0.3051  -0.0666  -0.2641  -0.3505  -0.2853  -0.1589  -0.1794  -0.0482
7  0.5206  -0.3552   0.6784   0.3225   0.0576  -0.1290   0.1037  -0.0854   0.0224
8  0.8319  -0.2848  -0.0071  -0.1094   0.0307  -0.0981  -0.0140   0.4485   0.0543
```

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.

```
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.FactorLoadingEstimationMethod =
            FactorAnalysis.Model.MaximumLikelihood;
        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());
    }
}
```

Output

```
Unique Error Variances
     0
```

```
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  0.2309

  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  -0.1528
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  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  0.0350
1  1.0000
2  7.1494
3  12.0000
4  0.8476
5  5.0000
```

FactorAnalysis.MatrixType Enumeration

Summary

Matrix type.

```
public enumeration Imsl.Stat.FactorAnalysis.MatrixType
```

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

Summary

Model type.

`public enumeration Imsl.Stat.FactorAnalysis.Model`

Fields

`AlphaFactorAnalysis`

`public Imsl.Stat.FactorAnalysis.Model 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`

`public Imsl.Stat.FactorAnalysis.Model GeneralizedLeastSquares`

Description

Indicates generalized least-squares (common factor model) method used to obtain the estimates.

`ImageFactorAnalysis`

`public Imsl.Stat.FactorAnalysis.Model ImageFactorAnalysis`

Description

Indicates Image-factor analysis (common factor model) method used to obtain the estimates.

`MaximumLikelihood`

`public Imsl.Stat.FactorAnalysis.Model MaximumLikelihood`

Description

Indicates maximum likelihood method used to obtain the estimates. Degrees of freedom is used for this estimation method.

`PrincipalComponent`

`public Imsl.Stat.FactorAnalysis.Model 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`

`public Imsl.Stat.FactorAnalysis.Model UnweightedLeastSquares`

Description

Indicates unweighted least-squares (common factor model) method used to obtain the estimates. This option is the default.

DiscriminantAnalysis Class

Summary

Performs a linear or a quadratic discriminant function analysis among several known groups and the use of either reclassification, split sample, or the leaving-out-one methods in order to evaluate the rule.

`public class Imsl.Stat.DiscriminantAnalysis`

Properties

ClassificationMethod

```
public Imsl.Stat.DiscriminantAnalysis.Classification ClassificationMethod
 {get; set; }
```

Description

The classification method.

Use `Classification` member `Reclassification` or `LeaveOutOne`.

By default, `Classification.Reclassification` is used.

CovarianceComputation

```
public Imsl.Stat.DiscriminantAnalysis.CovarianceMatrix
 CovarianceComputation {get; set; }
```

Description

The type of covariance matrices to be computed.

Use `CovarianceMatrix` class member `Pooled` or `PooledGroup`.

By default, `CovarianceMatrix.PooledGroup` is used.

DiscriminationMethod

```
public Imsl.Stat.DiscriminantAnalysis.Discrimination DiscriminationMethod
 {get; set; }
```

Description

The discrimination method.

Use `Discrimination` member `Linear` or `Quadratic`.

By default, `Discrimination.Linear` is used.

NRowsMissing

```
public int NRowsMissing {get; }
```

Description

Returns the number of rows of data encountered containing missing values (`NaN`).

If a row of data contains a missing value (`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 computed.

Use `PriorProbabilities` member `PriorEqual` or `PriorProportional`.

By default, `PriorProbabilities.PriorEqual` is used.

Constructor

DiscriminantAnalysis

```
public DiscriminantAnalysis(int nVariables, int nGroups)
```

Description

Constructor for `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

GetClassMembership

```
public int[] GetClassMembership()
```

Description

Returns the group number 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.

Returns

An `int` array containing the group to which the observation was classified.

GetClassTable

```
public double[,] GetClassTable()
```

Description

Returns the classification table.

Each observation that is classified and has a group number equal to 1.0, 2.0, ..., `nGroups` is entered into the table. The rows of the table correspond to the known group membership. The columns refer to the group to which the observation was classified.

Returns

A `nGroups` × `nGroups` `double` array containing the classification table.

GetCoefficients

```
public double[,] GetCoefficients()
```

Description

Returns the linear discriminant function coefficients.

The first column of the array contains the constant term, and the remaining columns contain the variable coefficients. The i-th row of the returned array corresponds to group i. The coefficients are always computed as linear discriminant function coefficients even when quadratic discrimination is specified.

Returns

A `double` array containing the linear discriminant function coefficients.

GetCovariance

```
public double[,,] GetCovariance()
```

Description

Returns the array of covariances.

Here, g `= nGroups + 1` unless pooled only covariance matrices are computed, in which case g `= 1`. 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.

Returns

A `nVariables`×`nVariables` ×g `double` array containing the covariances.

GetGroupCounts

```
public int[] GetGroupCounts()
```

Description

Returns the group counts.

Returns

An `int` array of length `nGroups` containing the number of observations in each group.

GetMahalanobis

```
public double[,] GetMahalanobis()
```

Description

Returns the Mahalanobis distances between the group means.

For linear discrimination, the Mahalanobis distance

$$D_{ij}^2$$

between group means i and j is computed using the within covariance matrix for group i in place of the pooled covariance matrix.

Returns

A `nGroups`×`nGroups` `double` array containing the Mahalanobis distances between the group means.

GetMeans

`public double[,] GetMeans()`

Description

Returns the variable means.

The i-th row of the returned array contains the group i variable means.

Returns

A `double` array containing the variable means.

GetPrior

`public double[] GetPrior()`

Description

Returns the prior probabilities for each group.

The elements of this vector should sum to 1.0. If this member function is not called, the elements are set so as to be equal if `PriorType` is set to `PriorProbabilities.PriorEqual` or they are set to be proportional to the sample size in each group if `PriorType` is set to `PriorProbabilities.PriorProportional`.

Returns

A `double` vector of length `nGroups` containing the prior probabilities for each group.

GetProbability

`public double[,] GetProbability()`

Description

Returns the posterior probabilities for each observation.

Returns

A `x.GetLength(0)`×`nGroups` `double` array containing the posterior probabilities for each observation.

GetStatistics

`public double[] GetStatistics()`

Description

Returns statistics.

i	Statistics[i]
0	Sum of the degrees of freedom for the within-covariance matrices.
1	Chi-squared statistic.
2	The degrees of freedom in the chi-squared statistic.
3	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).
4 thru 4 + `nGroups`	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.
Last `nGroups + 1` elements	Sum of the weights within each group.
Last element	Sum of the weights in all groups.

Returns

A `double` array containing output statistics.

SetPrior

`public void SetPrior(double[] prior)`

Description

Sets the prior probabilities for each group.

The elements of `prior` should sum to 1.0. If this member function is not called, the elements of `prior` are set so as to be equal if `PriorType` is set to `PriorProbabilities.PriorEqual` or they are set to be proportional to the sample size in each group if `PriorType` is set to `PriorProbabilities.PriorProportional`.

Parameter

`prior` – A `double` vector of length `nGroups` containing the prior probabilities for each group.

Update

`public void Update(double[,] x)`

Description

Processes a set of observations and performs a linear or quadratic discriminant function analysis among the several known groups.

The first `nVariables` columns correspond to the variables, and the last column (column `nVariables`) contains the group numbers. The groups must be numbered 1,2, ..., `nGroups`.

Parameter

`x` – A `double` matrix containing the observations.

`Imsl.Stat.SumOfWeightsNegException` id is thrown if the sum of the weights have become negative.

`Imsl.Stat.EmptyGroupException` id is thrown if there are no observations in a group. Cannot compute statistics.

`Imsl.Stat.CovarianceSingularException` id is thrown if the variance-Covariance matrix is singular.

`Imsl.Stat.PooledCovarianceSingularException` id is thrown if the pooled variance-Covariance matrix is singular.

Update

`public void Update(double[,] x, int groupIndex)`

Description

Processes a set of observations and performs a linear or quadratic discriminant function analysis among the several known groups.

The first `nVariables` columns correspond to the variables, excluding the `groupIndex` column. The groups must be numbered 1,2, ..., `nGroups`.

Parameters

`x` – A `double` matrix containing the observations.

`groupIndex` – An `int` containing the column index of `x` in which the group numbers are stored.

`Imsl.Stat.SumOfWeightsNegException` id is thrown if the sum of the weights have become negative.

`Imsl.Stat.EmptyGroupException` id is thrown if there are no observations in a group. Cannot compute statistics.

`Imsl.Stat.CovarianceSingularException` id is thrown if the variance-Covariance matrix is singular.

`Imsl.Stat.PooledCovarianceSingularException` id is thrown if the pooled variance-Covariance matrix is singular.

Update

`public void Update(double[,] x, int[] varIndex)`

Description

Processes a set of observations and performs a linear or quadratic discriminant function analysis among the several known groups.

The columns indicated in `varIndex` correspond to the variables, and the last column (column `nVariables`) contains the group numbers. The groups must be numbered 1,2, ..., `nGroups`.

Parameters

`x` – A `double` matrix containing the observations.

`varIndex` – An `int` array containing the column indices in `x` that correspond to the variables to be used in the analysis.

`Imsl.Stat.SumOfWeightsNegException` id is thrown if the sum of the weights have become negative.

`Imsl.Stat.EmptyGroupException` id is thrown if there are no observations in a group. Cannot compute statistics.

`Imsl.Stat.CovarianceSingularException` id is thrown if the variance-Covariance matrix is singular.

`Imsl.Stat.PooledCovarianceSingularException` id is thrown if the pooled variance-Covariance matrix is singular.

Update

`public void Update(double[,] x, double[] frequencies, double[] weights)`

Description

Processes a set of observations and associated frequencies and weights then performs a linear or quadratic discriminant function analysis among the several known groups.

The first `nVariables` columns correspond to the variables, and the last column (column `nVariables`) contains the group numbers. The groups must be numbered 1,2, ..., `nGroups`.

Parameters

`x` – A `double` matrix containing the observations.

`frequencies` – A `double` array containing the associated frequencies.

`weights` – A `double` array containing the associated weights.

`Imsl.Stat.SumOfWeightsNegException` id is thrown if the sum of the weights have become negative.

`Imsl.Stat.EmptyGroupException` id is thrown if there are no observations in a group. Cannot compute statistics.

`Imsl.Stat.CovarianceSingularException` id is thrown if the variance-Covariance matrix is singular.

`Imsl.Stat.PooledCovarianceSingularException` id is thrown if the pooled variance-Covariance matrix is singular.

Update

`public void Update(double[,] x, int groupIndex, int[] varIndex)`

Description

Processes a set of observations and performs a linear or quadratic discriminant function analysis among the several known groups.

The columns indicated in `varIndex` correspond to the variables, and `groupIndex` column contains the group numbers. The groups must be numbered 1,2, ..., `nGroups`.

Parameters

`x` – A `double` matrix containing the observations.

`groupIndex` – An `int` containing the column index of `x` in which the group numbers are stored.

`varIndex` – An `int` array containing the column indices in `x` that correspond to the variables to be used in the analysis.

`Imsl.Stat.SumOfWeightsNegException` id is thrown if the sum of the weights have become negative.

`Imsl.Stat.EmptyGroupException` id is thrown if there are no observations in a group. Cannot compute statistics.

`Imsl.Stat.CovarianceSingularException` id is thrown if the variance-Covariance matrix is singular.

`Imsl.Stat.PooledCovarianceSingularException` id is thrown if the pooled variance-Covariance matrix is singular.

Update

```
public void Update(double[,] x, int groupIndex, double[] frequencies,
  double[] weights)
```

Description

Processes a set of observations and associated frequencies and weights then performs a linear or quadratic discriminant function analysis among the several known groups.

The first `nVariables` columns correspond to the variables, excluding the `groupIndex` column. The groups must be numbered 1,2, ..., `nGroups`.

Parameters

`x` – A `double` matrix containing the observations.

`groupIndex` – An `int` containing the column index of `x` in which the group numbers are stored.

`frequencies` – A `double` array containing the associated frequencies.

`weights` – A `double` array containing the associated weights.

`Imsl.Stat.SumOfWeightsNegException` id is thrown if the sum of the weights have become negative.

`Imsl.Stat.EmptyGroupException` id is thrown if there are no observations in a group. Cannot compute statistics.

`Imsl.Stat.CovarianceSingularException` id is thrown if the variance-Covariance matrix is singular.

`Imsl.Stat.PooledCovarianceSingularException` id is thrown if the pooled variance-Covariance matrix is singular.

Update

```
public void Update(double[,] x, int[] varIndex, double[] frequencies,
  double[] weights)
```

Description

Processes a set of observations and associated frequencies and weights then performs a linear or quadratic discriminant function analysis among the several known groups.

The columns indicated in `varIndex` correspond to the variables, and the last column (column `nVariables`) contains the group numbers. The groups must be numbered 1,2, ..., `nGroups`.

Parameters

`x` – A `double` matrix containing the observations.

`varIndex` – An `int` array containing the column indices in `x` that correspond to the variables to be used in the analysis.

`frequencies` – A `double` array containing the associated frequencies.

`weights` – A `double` array containing the associated weights.

`Imsl.Stat.SumOfWeightsNegException` id is thrown if the sum of the weights have become negative.

`Imsl.Stat.EmptyGroupException` id is thrown if there are no observations in a group. Cannot compute statistics.

`Imsl.Stat.CovarianceSingularException` id is thrown if the variance-Covariance matrix is singular.

`Imsl.Stat.PooledCovarianceSingularException` id is thrown if the pooled variance-Covariance matrix is singular.

Update

```
public void Update(double[,] x, int groupIndex, int[] varIndex, double[]
  frequencies, double[] weights)
```

Description

Processes a set of observations and associated frequencies and weights then performs a linear or quadratic discriminant function analysis among the several known groups.

The columns indicated in `varIndex` correspond to the variables, and `groupIndex` column contains the group numbers. The groups must be numbered 1,2, ..., `nGroups`.

Parameters

`x` – A `double` matrix containing the observations.

`groupIndex` – An `int` containing the column index of `x` in which the group numbers are stored.

`varIndex` – An `int` array containing the column indices in `x` that correspond to the variables to be used in the analysis.

`frequencies` – A `double` array containing the associated frequencies.

`weights` – A `double` array containing the associated weights.

`Imsl.Stat.SumOfWeightsNegException` id is thrown if the sum of the weights have become negative.

`Imsl.Stat.EmptyGroupException` id is thrown if there are no observations in a group. Cannot compute statistics.

`Imsl.Stat.CovarianceSingularException` id is thrown if the variance-Covariance matrix is singular.

`Imsl.Stat.PooledCovarianceSingularException` id is thrown if the pooled variance-Covariance matrix is singular.

Description

Class `DiscriminantAnalysis` performs discriminant function analysis using either linear or quadratic discrimination. The output from `DiscriminantAnalysis` includes a measure of distance between the groups, a table summarizing the classification results, a matrix containing the posterior probabilities of group membership for each observation, and the within-sample means and covariance matrices. The linear discriminant function coefficients are also computed.

All observations are input during one call to `DiscriminantAnalysis`, a method of operation that has the advantage of simplicity.

All observations in `x` are used to compute the means. The covariance matrices are factored. Requested statistics of interest are computed: the linear discriminant functions, the prior probabilities, the log of the determinant of each of the covariance matrices, a test statistic for testing that all of the within-group covariance matrices are equal, and a matrix of Mahalanobis distances between the groups. The matrix of Mahalanobis distances 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,

$$S_i = \frac{1}{N_i - 1} \sum_{j=1}^{M_i} w_j f_j (x_j - \overline{x})(x_j - \overline{x})^T$$

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 $\overline{x}$ denotes the mean vector of the observations in group i. The mean vectors are computed as

$$\overline{x} = \frac{1}{W_i} \sum_{j=1}^{M_i} w_j f_j x_j$$

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:

$$z_i = ln(p_i) - 0.5\overline{x_i}^T S_p^{-1} \overline{x_i} + x^T S_p^{-1} \overline{x_i}$$

where $ln(pi)$ 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:

$$D_{ij}^2 = (\overline{x_i} - \overline{x_j})^T S^{-1} (\overline{x_i} - \overline{x_j})$$

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

$$C^{-1} = 1 - \frac{2p^2 + 3p - 1}{6(p+1)(k-1)} \left(\sum_{i=1}^{k} \frac{1}{n_i} - \frac{1}{\Sigma_j n_j} \right)$$

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

$$\hat{q}_i(x) = \frac{e^{-\frac{1}{2}D_i^2(x)}}{\sum_{j=1}^{k} e^{-\frac{1}{2}D_j^2(x)}}$$

where

$$D_i^2(x) = \begin{cases} (x - \overline{x_i})^T S_i^{-1} (x - \overline{x_i}) + ln\,|S_i| - 2ln(p_i) & LINEAR\ or\ QUADRATIC \\ (x - \overline{x_i})^T S_p^{-1} (x - \overline{x_i}) - 2ln(p_i) & LINEAR\ POOLED \end{cases}$$

For the leaving-out-one method of classification, the sample mean vector and sample covariance matrices in the formula for

$$D_i^2(x)$$

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 observations in X is classified into a group; the result is tabulated in the matrix `CLASS` and saved in the vector `ICLASS`. `CLASS` is not altered at this stage if `X(i, IGRP)` contains a group number that is out of range. If the reclassification method is specified, then all observations with no missing values in the `nVariables` classification variables 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 `CLASS` 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 `X(i, IWT)`, if $IWT > 0$ and a frequency of 1.0. See Lachenbruch (1975, page 36) for the required adjustment.

Finally, upon completion, the covariance matrices are computed from their LU factorizations.

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.

```
using System;
using Imsl.Stat;
using PrintMatrix = Imsl.Math.PrintMatrix;

public class DiscriminantAnalysisEx1
{
    public static void  Main(String[] args)
    {
        double[,] xorig = {
            {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},
{1.0, 4.9, 3.6, 1.4, .1},
{1.0, 4.4, 3.0, 1.3, .2},
{1.0, 5.1, 3.4, 1.5, .2},
{1.0, 5.0, 3.5, 1.3, .3},
{1.0, 4.5, 2.3, 1.3, .3},
{1.0, 4.4, 3.2, 1.3, .2},
{1.0, 5.0, 3.5, 1.6, .6},
{1.0, 5.1, 3.8, 1.9, .4},
{1.0, 4.8, 3.0, 1.4, .3},
{1.0, 5.1, 3.8, 1.6, .2},
{1.0, 4.6, 3.2, 1.4, .2},
{1.0, 5.3, 3.7, 1.5, .2},
{1.0, 5.0, 3.3, 1.4, .2},
{2.0, 7.0, 3.2, 4.7, 1.4},
{2.0, 6.4, 3.2, 4.5, 1.5},
{2.0, 6.9, 3.1, 4.9, 1.5},
{2.0, 5.5, 2.3, 4.0, 1.3},
{2.0, 6.5, 2.8, 4.6, 1.5},
{2.0, 5.7, 2.8, 4.5, 1.3},
{2.0, 6.3, 3.3, 4.7, 1.6},
{2.0, 4.9, 2.4, 3.3, 1.0},
{2.0, 6.6, 2.9, 4.6, 1.3},
{2.0, 5.2, 2.7, 3.9, 1.4},
{2.0, 5.0, 2.0, 3.5, 1.0},
{2.0, 5.9, 3.0, 4.2, 1.5},
{2.0, 6.0, 2.2, 4.0, 1.0},
{2.0, 6.1, 2.9, 4.7, 1.4},
{2.0, 5.6, 2.9, 3.6, 1.3},
{2.0, 6.7, 3.1, 4.4, 1.4},
{2.0, 5.6, 3.0, 4.5, 1.5},
{2.0, 5.8, 2.7, 4.1, 1.0},
{2.0, 6.2, 2.2, 4.5, 1.5},
```

```
{2.0, 5.6, 2.5, 3.9, 1.1},
{2.0, 5.9, 3.2, 4.8, 1.8},
{2.0, 6.1, 2.8, 4.0, 1.3},
{2.0, 6.3, 2.5, 4.9, 1.5},
{2.0, 6.1, 2.8, 4.7, 1.2},
{2.0, 6.4, 2.9, 4.3, 1.3},
{2.0, 6.6, 3.0, 4.4, 1.4},
{2.0, 6.8, 2.8, 4.8, 1.4},
{2.0, 6.7, 3.0, 5.0, 1.7},
{2.0, 6.0, 2.9, 4.5, 1.5},
{2.0, 5.7, 2.6, 3.5, 1.0},
{2.0, 5.5, 2.4, 3.8, 1.1},
{2.0, 5.5, 2.4, 3.7, 1.0},
{2.0, 5.8, 2.7, 3.9, 1.2},
{2.0, 6.0, 2.7, 5.1, 1.6},
{2.0, 5.4, 3.0, 4.5, 1.5},
{2.0, 6.0, 3.4, 4.5, 1.6},
{2.0, 6.7, 3.1, 4.7, 1.5},
{2.0, 6.3, 2.3, 4.4, 1.3},
{2.0, 5.6, 3.0, 4.1, 1.3},
{2.0, 5.5, 2.5, 4.0, 1.3},
{2.0, 5.5, 2.6, 4.4, 1.2},
{2.0, 6.1, 3.0, 4.6, 1.4},
{2.0, 5.8, 2.6, 4.0, 1.2},
{2.0, 5.0, 2.3, 3.3, 1.0},
{2.0, 5.6, 2.7, 4.2, 1.3},
{2.0, 5.7, 3.0, 4.2, 1.2},
{2.0, 5.7, 2.9, 4.2, 1.3},
{2.0, 6.2, 2.9, 4.3, 1.3},
{2.0, 5.1, 2.5, 3.0, 1.1},
{2.0, 5.7, 2.8, 4.1, 1.3},
{3.0, 6.3, 3.3, 6.0, 2.5},
{3.0, 5.8, 2.7, 5.1, 1.9},
{3.0, 7.1, 3.0, 5.9, 2.1},
{3.0, 6.3, 2.9, 5.6, 1.8},
{3.0, 6.5, 3.0, 5.8, 2.2},
{3.0, 7.6, 3.0, 6.6, 2.1},
{3.0, 4.9, 2.5, 4.5, 1.7},
{3.0, 7.3, 2.9, 6.3, 1.8},
{3.0, 6.7, 2.5, 5.8, 1.8},
{3.0, 7.2, 3.6, 6.1, 2.5},
{3.0, 6.5, 3.2, 5.1, 2.0},
{3.0, 6.4, 2.7, 5.3, 1.9},
{3.0, 6.8, 3.0, 5.5, 2.1},
{3.0, 5.7, 2.5, 5.0, 2.0},
{3.0, 5.8, 2.8, 5.1, 2.4},
{3.0, 6.4, 3.2, 5.3, 2.3},
{3.0, 6.5, 3.0, 5.5, 1.8},
{3.0, 7.7, 3.8, 6.7, 2.2},
{3.0, 7.7, 2.6, 6.9, 2.3},
{3.0, 6.0, 2.2, 5.0, 1.5},
{3.0, 6.9, 3.2, 5.7, 2.3},
{3.0, 5.6, 2.8, 4.9, 2.0},
{3.0, 7.7, 2.8, 6.7, 2.0},
{3.0, 6.3, 2.7, 4.9, 1.8},
{3.0, 6.7, 3.3, 5.7, 2.1},
```

```
    {3.0, 7.2, 3.2, 6.0, 1.8},
    {3.0, 6.2, 2.8, 4.8, 1.8},
    {3.0, 6.1, 3.0, 4.9, 1.8},
    {3.0, 6.4, 2.8, 5.6, 2.1},
    {3.0, 7.2, 3.0, 5.8, 1.6},
    {3.0, 7.4, 2.8, 6.1, 1.9},
    {3.0, 7.9, 3.8, 6.4, 2.0},
    {3.0, 6.4, 2.8, 5.6, 2.2},
    {3.0, 6.3, 2.8, 5.1, 1.5},
    {3.0, 6.1, 2.6, 5.6, 1.4},
    {3.0, 7.7, 3.0, 6.1, 2.3},
    {3.0, 6.3, 3.4, 5.6, 2.4},
    {3.0, 6.4, 3.1, 5.5, 1.8},
    {3.0, 6.0, 3.0, 4.8, 1.8},
    {3.0, 6.9, 3.1, 5.4, 2.1},
    {3.0, 6.7, 3.1, 5.6, 2.4},
    {3.0, 6.9, 3.1, 5.1, 2.3},
    {3.0, 5.8, 2.7, 5.1, 1.9},
    {3.0, 6.8, 3.2, 5.9, 2.3},
    {3.0, 6.7, 3.3, 5.7, 2.5},
    {3.0, 6.7, 3.0, 5.2, 2.3},
    {3.0, 6.3, 2.5, 5.0, 1.9},
    {3.0, 6.5, 3.0, 5.2, 2.0},
    {3.0, 6.2, 3.4, 5.4, 2.3},
    {3.0, 5.9, 3.0, 5.1, 1.8}
};

int[] ipermu = new int[]{2, 3, 4, 5, 1};

double[,] x = new double[xorig.GetLength(0),xorig.GetLength(1)];

for (int i = 0; i < xorig.GetLength(0); i++)
{
    for (int j = 1; j < xorig.GetLength(1); j++)
    {
        x[i,j - 1] = xorig[i,j];
    }
}
for (int i = 0; i < xorig.GetLength(0); i++)
{
    x[i,4] = xorig[i,0];
}

int nvar = x.GetLength(1) - 1;

DiscriminantAnalysis da = new DiscriminantAnalysis(nvar, 3);
da.CovarianceComputation =
    Imsl.Stat.DiscriminantAnalysis.CovarianceMatrix.Pooled;
da.ClassificationMethod =
Imsl.Stat.DiscriminantAnalysis.Classification.Reclassification;

da.Update(x);
new PrintMatrix("Xmean are:  ").SetPageWidth(80).Print(da.GetMeans());
new PrintMatrix("Coef:  ").SetPageWidth(80).Print(da.GetCoefficients());
new PrintMatrix("Counts: ").SetPageWidth(80).Print(da.GetGroupCounts());
new PrintMatrix("Stats: ").SetPageWidth(80).Print(da.GetStatistics());
```

```
        new PrintMatrix("ClassMembership: ").SetPageWidth(80).Print(da.GetClassMembership());
        new PrintMatrix("ClassTable: ").SetPageWidth(80).Print(da.GetClassTable());
        double[,,] cov = da.GetCovariance();
        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 + " : ").SetPageWidth(80).Print(tmpCov);
        }
        new PrintMatrix("Prior : ").SetPageWidth(80).Print(da.GetPrior());
        new PrintMatrix("PROB: ").SetPageWidth(80).Print(da.GetProbability());
        new PrintMatrix("MAHALANOBIS: ").SetPageWidth(80).Print(da.GetMahalanobis());
        Console.Out.WriteLine("nrmiss = " + da.NRowsMissing);
    }
}
```

Output

```
         Xmean are:
     0      1      2      3
0  5.006  3.428  1.462  0.246
1  5.936  2.77   4.26   1.326
2  6.588  2.974  5.552  2.026

                                    Coef:
          0                  1                  2                  3
0   -86.308469973674   23.5441667229203  23.5878704955898   -16.4306390229439
1   -72.8526074006422  15.6982090760379   7.07250983729562    5.21145093416415
2  -104.36831998645    12.4458489937766   3.68527961207532   12.7665449735348

          4
0  -17.3984107815644
1    6.43422920040657
2   21.0791130134185

Counts:
   0
0  50
1  50
2  50

        Stats:
            0
 0  147
 1    NaN
 2    NaN
 3    NaN
 4    NaN
 5    NaN
 6    NaN
```

```
 7   -9.95853877004797
 8   50
 9   50
10   50
11  150

ClassMembership:
     0
  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
 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   2
 53   2
 54   2
 55   2
 56   2
 57   2
 58   2
 59   2
 60   2
 61   2
 62   2
 63   2
 64   2
 65   2
 66   2
 67   2
 68   2
 69   2
 70   3
 71   2
 72   2
 73   2
 74   2
 75   2
 76   2
 77   2
 78   2
 79   2
 80   2
 81   2
 82   2
 83   3
 84   2
 85   2
 86   2
 87   2
 88   2
 89   2
 90   2
 91   2
 92   2
 93   2
 94   2
 95   2
 96   2
 97   2
 98   2
 99   2
100   3
101   3
102   3
103   3
```

```
104  3
105  3
106  3
107  3
108  3
109  3
110  3
111  3
112  3
113  3
114  3
115  3
116  3
117  3
118  3
119  3
120  3
121  3
122  3
123  3
124  3
125  3
126  3
127  3
128  3
129  3
130  3
131  3
132  3
133  2
134  3
135  3
136  3
137  3
138  3
139  3
140  3
141  3
142  3
143  3
144  3
145  3
146  3
147  3
148  3
149  3

 ClassTable:
   0   1   2
0  50   0   0
1   0  48   2
2   0   1  49

                    Covariance Matrix 0 :
           0                    1                    2
0  0.265008163265306   0.0927210884353742  0.167514285714286
```

```
1  0.0927210884353742  0.115387755102041   0.055243537414966
2  0.167514285714286   0.055243537414966   0.185187755102041
3  0.0384013605442177  0.0327102040816327  0.042665306122449

           3
0  0.0384013605442177
1  0.0327102040816327
2  0.042665306122449
3  0.0418816326530612

       Prior :
           0
0  0.333333333333333
1  0.333333333333333
2  0.333333333333333

                             PROB:
             0                     1                     2
  0  1                     3.89635792768677E-22  2.61116827494833E-42
  1  1                     7.21796991863879E-18  5.04214334588401E-37
  2  1                     1.46384894952907E-19  4.67593159333071E-39
  3  1                     1.26853637674403E-16  3.56661049202016E-35
  4  1                     1.63738744612726E-22  1.08260526717561E-42
  5  1                     3.88328166174543E-21  4.56654013405467E-40
  6  1                     1.1134694458599E-18   2.3026084834884E-37
  7  1                     3.87758637727045E-20  1.07449600387617E-39
  8  0.999999999999998     1.90281305967755E-15  9.48293561788352E-34
  9  1                     1.11180260918759E-18  2.72405964325484E-38
 10  1                     1.18527748898975E-23  3.23708368191298E-44
 11  1                     1.62164851137697E-18  1.83320074038366E-37
 12  1                     1.45922504711622E-18  3.2625064352377E-38
 13  1                     1.11721885779029E-19  1.31664193135497E-39
 14  1                     5.4873987251784E-30   1.53126472959902E-52
 15  1                     1.26150509583788E-27  2.26870462780447E-48
 16  1                     6.75433806261566E-25  3.86827125184469E-45
 17  1                     4.22374070046694E-21  1.22431307255763E-40
 18  1                     1.77491130351548E-22  2.5521532433363E-42
 19  1                     2.59323737921836E-22  5.79207874344749E-42
 20  1                     1.27463865682517E-19  4.35777421418678E-39
 21  1                     1.4659990076799E-20   1.98724138647432E-39
 22  1                     6.56928044945199E-25  7.76917736630943E-46
 23  0.999999999999991     8.91234785423208E-15  9.1786241650176E-32
 24  0.999999999999999     1.07070246199648E-15  1.16751587102608E-33
 25  1                     2.49733903598925E-16  5.71026880713927E-35
 26  1                     3.96773183597681E-17  4.37862393400249E-35
 27  1                     1.54816504878351E-21  1.59535976667668E-41
 28  1                     9.27184652389738E-22  6.29795546615504E-42
 29  1                     9.66514422364528E-17  2.97797411204608E-35
 30  1                     2.29993587694263E-16  7.18266552062749E-35
 31  1                     1.97540361007101E-19  2.78833402097428E-38
 32  1                     7.10004097342709E-27  2.21640831858024E-48
 33  1                     1.61029483654946E-28  2.74378339740563E-50
 34  1                     1.20521934033381E-17  1.2772450797832E-36
 35  1                     1.59718567904273E-21  9.03377178189315E-42
 36  1                     1.93986888092869E-24  1.66280764122895E-45
 37  1                     3.31023376400147E-23  7.00497072116312E-44
```

38	1	4.19024193534821E-17	6.99144060854016E-36
39	1	1.76935863474539E-20	3.54169362585279E-40
40	1	1.06301363512176E-21	2.00386616263149E-41
41	0.999999999978258	2.17421702175081E-11	1.21378079456305E-28
42	1	1.54075327236441E-18	1.30571860833262E-37
43	0.999999999999999	8.94058875293973E-16	1.3155106225373E-32
44	1	1.61620622204115E-17	3.2059920592081E-35
45	1	1.71474317216017E-16	7.17243513417258E-35
46	1	2.0830893288107E-22	2.28978349710803E-42
47	1	2.7934821528124E-18	2.62953861900424E-37
48	1	2.59756035567857E-23	9.82081977684015E-44
49	1	2.32225794022775E-20	4.24175670110456E-40
50	1.96973175506613E-18	0.999889412240982	0.000110587759018098
51	1.24287799621613E-19	0.999257470339862	0.000742529660138549
52	2.0882630542231E-22	0.995806947154067	0.00419305284593237
53	2.19889843940163E-22	0.999642349804226	0.000357650195773665
54	4.21367813304238E-23	0.995590345144127	0.00440965485587324
55	8.12728653249083E-23	0.99850201836847	0.00149798163153038
56	3.54989967813166E-22	0.98583457962357	0.0141654203764301
57	5.00706455445538E-14	0.999999888018824	1.11981125915622E-07
58	5.68333389389098E-20	0.999878135052759	0.000121864947241113
59	1.24103857349892E-20	0.999502691481115	0.000497308518885132
60	1.95662763937994E-18	0.99999857915944	1.42084056032926E-06
61	5.96890036424978E-20	0.999229428361292	0.000770571638708534
62	2.71612817142458E-18	0.999998779830561	1.22016943887906E-06
63	1.18444452318768E-23	0.994326714395047	0.00567328560495259
64	5.57493127051318E-14	0.999998350784465	1.64921547935997E-06
65	2.36951149493997E-17	0.999957317877693	4.26821223073998E-05
66	8.42932810347787E-24	0.980647108438011	0.0193528915619887
67	2.50507161487087E-16	0.999999084828366	9.15171633415319E-07
68	1.67035240315192E-27	0.959573472466896	0.0404265275331035
69	1.34150265115522E-17	0.999996703894565	3.29610543460692E-06
70	7.40811758162493E-28	0.253228224738174	0.746771775261826
71	9.39929180687634E-17	0.999990654708731	9.34529126858083E-06
72	7.67467217173111E-29	0.815532827469118	0.184467172530882
73	2.68301817862459E-22	0.999572253144266	0.000427746855734505
74	7.81387455762235E-18	0.999975785420659	2.42145793413337E-05
75	2.07320734549583E-18	0.999917094703006	8.29052969937477E-05
76	6.35753788317195E-23	0.998254064057654	0.00174593594234629
77	5.63947317748202E-27	0.6892131192651	0.3107868807349
78	3.77352772315036E-23	0.992516862421269	0.00748313757873124
79	9.55533837753134E-12	0.999999980884202	1.91062427723588E-08
80	1.02210867392728E-17	0.999996992252333	3.0077476672767E-06
81	9.64807489487809E-16	0.999999673329606	3.26670393036477E-07
82	1.6164048498996E-16	0.9999962215592	3.778440799605E-06
83	4.24195194474068E-32	0.143391908078749	0.856608091921251
84	1.72451373302881E-24	0.963557581487526	0.0364424185124737
85	1.34474562081143E-20	0.994040068728999	0.00595993127100056
86	3.30486831694226E-21	0.998222327554005	0.00177767244599476
87	2.03457104837351E-23	0.99945569039693	0.000544309603069924
88	5.80698628888408E-18	0.999948628991219	5.13710087811031E-05
89	5.98119018015315E-21	0.999818313010677	0.000181686989322851
90	5.87861351190954E-23	0.999385580036369	0.000614419963631344
91	5.39900622927546E-22	0.99809340863166	0.00190659136833998
92	3.55950706996344E-18	0.999988714300912	1.12856990882795E-05
93	2.10414566195034E-14	0.999999886498331	1.13501647389283E-07

94	4.70087713218134E-21	0.999697977427644	0.000302022572355453
95	1.58432826245531E-17	0.999981736726967	1.8263273032574E-05
96	2.80229312743823E-19	0.999889168510548	0.000110831489452329
97	1.62691766654364E-18	0.999953595115214	4.64048847857305E-05
98	7.63837759915641E-11	0.9999999812503	1.86733161738005E-08
99	4.67930110528286E-19	0.999926941365602	7.30586343982778E-05
100	7.50307535787394E-52	7.12730304524431E-09	0.999999992872697
101	5.21380197778188E-38	0.00107825098684016	0.99892174901316
102	1.23126361178618E-42	2.5928263674499E-05	0.999974071736326
103	1.5374987093819E-38	0.00106813895788396	0.998931861042116
104	6.24250059805309E-46	1.81296336402271E-06	0.999998187036636
105	4.20928142264626E-49	6.65626292581717E-07	0.999999334373707
106	3.79783718920213E-33	0.0486202537563576	0.951379746243642
107	1.35217625957575E-42	0.000139546311732453	0.999860453688268
108	1.32338962875972E-42	0.000223531291267257	0.999776468708733
109	3.45335796991583E-46	1.72727719755284E-07	0.99999982727228
110	5.45266025357921E-32	0.0130535277071713	0.986946472292829
111	1.18256006692676E-37	0.00167387461672592	0.998326125383274
112	5.20432100382649E-39	0.000200635233279198	0.999799364766721
113	1.26995255164363E-40	0.000194867232288328	0.999805132767712
114	1.6853613804622E-45	1.00045460233037E-06	0.999998999545398
115	5.1416398277959E-40	2.60549340476461E-05	0.999973945065952
116	1.90982041288985E-35	0.00608355276834907	0.993916447231651
117	1.20779857988563E-44	1.50379915614154E-06	0.999998496200844
118	3.18126528330457E-59	1.31727867517054E-09	0.999999998682721
119	1.59851089004605E-33	0.220798984305311	0.779201015694688
120	1.11946077288319E-42	6.45186467453254E-06	0.999993548135325
121	3.03817020493178E-37	0.000827267586982933	0.999172732413017
122	6.03287894248635E-50	9.50983817394636E-07	0.999999049016183
123	1.9512605095192E-31	0.0971194197474795	0.90288058025252
124	1.95640815619155E-39	8.83684525244587E-05	0.999911631547476
125	1.10933653717363E-36	0.002679309662425	0.997320690337575
126	7.84199684983311E-30	0.188367543357746	0.811632456642254
127	7.96469039198895E-30	0.134243091251227	0.865756908748773
128	6.19064114103918E-44	1.30368066299741E-05	0.99998696319337
129	1.40644845645161E-32	0.103682284629142	0.896317715370858
130	4.10812925130498E-42	0.000144233750964738	0.999855766249035
131	1.55569699085988E-36	0.000519804735082123	0.999480195264918
132	1.32032958960942E-45	3.01409095623433E-06	0.999996985909044
133	1.28389062432083E-28	0.729388128031784	0.270611871968216
134	1.92656005411401E-35	0.0660225289487927	0.933977471051207
135	1.27108275807441E-45	2.15281844850908E-06	0.999997847181551
136	3.03896326390101E-44	8.88185881285476E-07	0.999999111814119
137	4.60597294289873E-35	0.00616564821325895	0.993834351786741
138	4.53863396078092E-29	0.192526178705555	0.807473821294445
139	2.14023245437832E-36	0.000829089537724701	0.999170910462275
140	6.57090159803634E-45	1.18080976021024E-06	0.99999881919024
141	6.20258777948126E-36	0.000427639823558763	0.999572360176441
142	5.21380197778188E-38	0.00107825098684016	0.99892174901316
143	1.07394549657031E-45	1.02851888652808E-06	0.999998971481114
144	4.04824911953177E-46	2.52498398896306E-07	0.999999747501601
145	4.97006952429139E-39	7.47336051808618E-05	0.999925266394819
146	4.61661069188626E-36	0.00589878415239281	0.994101215847607
147	5.54896239081565E-35	0.00314587355706823	0.996854126442932
148	1.61368724215064E-40	1.25746799233817E-05	0.999987425320077
149	2.85801160733409E-33	0.017542290775785	0.982457709224215

```
                      MAHALANOBIS:
           0                  1                  2
0    0                89.8641855820738  179.384712514278
1   89.8641855820738   0                 17.201066428396
2  179.384712514278   17.201066428396     0

nrmiss = 0
```

DiscriminantAnalysis.Discrimination Enumeration

Summary

Discrimination Methods.

`public enumeration Imsl.Stat.DiscriminantAnalysis.Discrimination`

Fields

`Linear`
`public Imsl.Stat.DiscriminantAnalysis.Discrimination Linear`

Description

Indicates a linear discrimination method.

`Quadratic`
`public Imsl.Stat.DiscriminantAnalysis.Discrimination Quadratic`

Description

Indicates a quadratic discrimination method.

DiscriminantAnalysis.CovarianceMatrix Enumeration

Summary

Covariance Matrix type.

`public enumeration Imsl.Stat.DiscriminantAnalysis.CovarianceMatrix`

Fields

`Pooled`
`public Imsl.Stat.DiscriminantAnalysis.CovarianceMatrix Pooled`

Description

Indicates Pooled covariances computed.

`PooledGroup`
`public Imsl.Stat.DiscriminantAnalysis.CovarianceMatrix PooledGroup`

Description

Indicates Pooled, group covariances computed.

DiscriminantAnalysis.Classification Enumeration

Summary

Classification Method.

`public enumeration Imsl.Stat.DiscriminantAnalysis.Classification`

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

Summary

Prior probabilities type.

```
public enumeration Imsl.Stat.DiscriminantAnalysis.PriorProbabilities
```

Fields

```
PriorEqual
public Imsl.Stat.DiscriminantAnalysis.PriorProbabilities PriorEqual
```

Description

Indicates prior probability type is to be prior equal.

```
PriorProportional
public Imsl.Stat.DiscriminantAnalysis.PriorProbabilities PriorProportional
```

Description

Indicates prior probability type is to be prior proportional.

Chapter 20: Probability Distribution Functions and Inverses

Types

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` and `Cdf.InverseNormal`, 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

$$F(x) = \mathrm{Prob}(X \leq x)$$

where $\mathrm{Prob}(\cdot)$ denotes the probability of an event. The distribution function is often called the *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` 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.

Discrete Random Variables

For discrete distributions, the function giving the probability that the random variable takes on specific values is called the *probability function*, defined by

$$p(x) = \mathrm{Prob}(X = x)$$

The CDF for a discrete random variable is

$$F(x) = \sum_{A} p(k)$$

where A is set such that $k \le x$. Since the distribution function is a step function, its inverse does not exist uniquely.

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 *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

$$\mathrm{Prob}(a \le X \le b) = \int_a^b f(x)\,dx$$

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 "Inverse" methods in the `Cdf` class compute the inverses of the distribution functions. That is, given F(`x`) (called "prob" for "probability"), a method such as, `InverseBeta` in the `Cdf` 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 `Beta` at 0.7, 0.999953 is obtained. A more precise result is obtained by evaluating `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

Summary

Cumulative probability distribution functions, probability density functions, and their inverses.

```
public class Imsl.Stat.Cdf
```

Methods

Beta

```
static public double Beta(double x, double pin, double qin)
```

Description

Evaluates the beta cumulative probability distribution function.

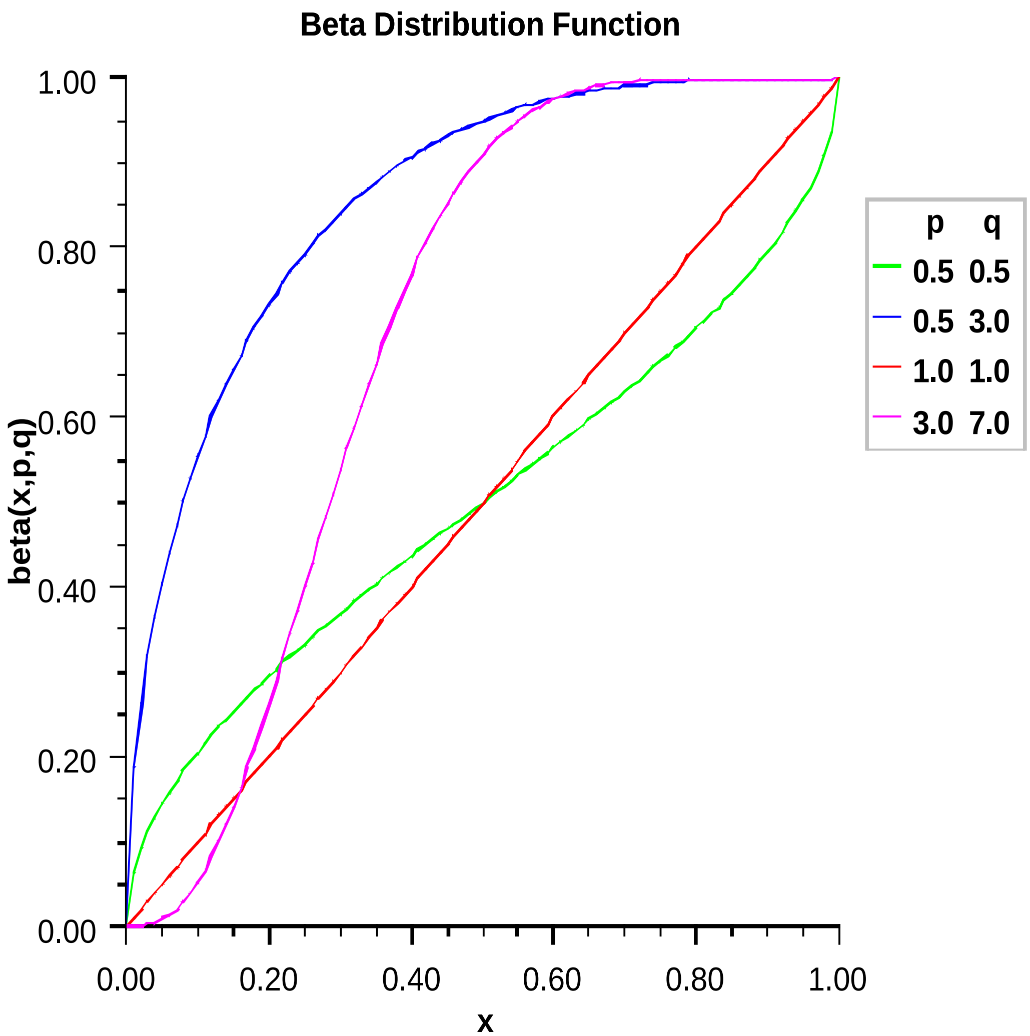
Method `Beta` evaluates the distribution function of a beta random variable with parameters `pin` and `qin`. This function is sometimes called the *incomplete beta ratio* and, with $p = pin$ and $q = qin$, is denoted by $I_x(p, q)$. It is given by

$$I_x\left(p,\, q\right) = \frac{\Gamma\left(p\right)\Gamma\left(q\right)}{\Gamma\left(p+q\right)} \int_0^x t^{p-1}\left(1-t\right)^{q-1} dt$$

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 $\beta_x(p,q)$.

`Beta` uses the method of Bosten and Battiste (1974).



Parameters

`x` – A `double` specifying the argument at which the function is to be evaluated.

`pin` – A `double` specifying the first beta distribution parameter.

`qin` – A `double` specifying the second beta distribution parameter.

Returns

A `double` specifying the probability that a beta random variable takes on a value less than or equal to x.

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.

BetaProb

`static public double BetaProb(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`.

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 p)
```

Description

Evaluates the binomial cumulative probability distribution function.

Method `Binomial` evaluates the distribution function of a binomial random variable with parameters n and p. 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, ε, is used as the starting value for summing 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 $p = 0$, `Binomial` is set to 1; and for the case $p = 1$, `Binomial` is set to 1 if $k = n$ and to 0 otherwise.

Parameters

k – An `int` specifying the argument for which the binomial distribution function is to be evaluated.

n – An `int` specifying the number of Bernoulli trials.

p – A `double` specifying the probability of success on each trial.

Returns

A `double` specifying 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 p probability of success.

BinomialProb

```
static public double BinomialProb(int k, int n, double p)
```

Description

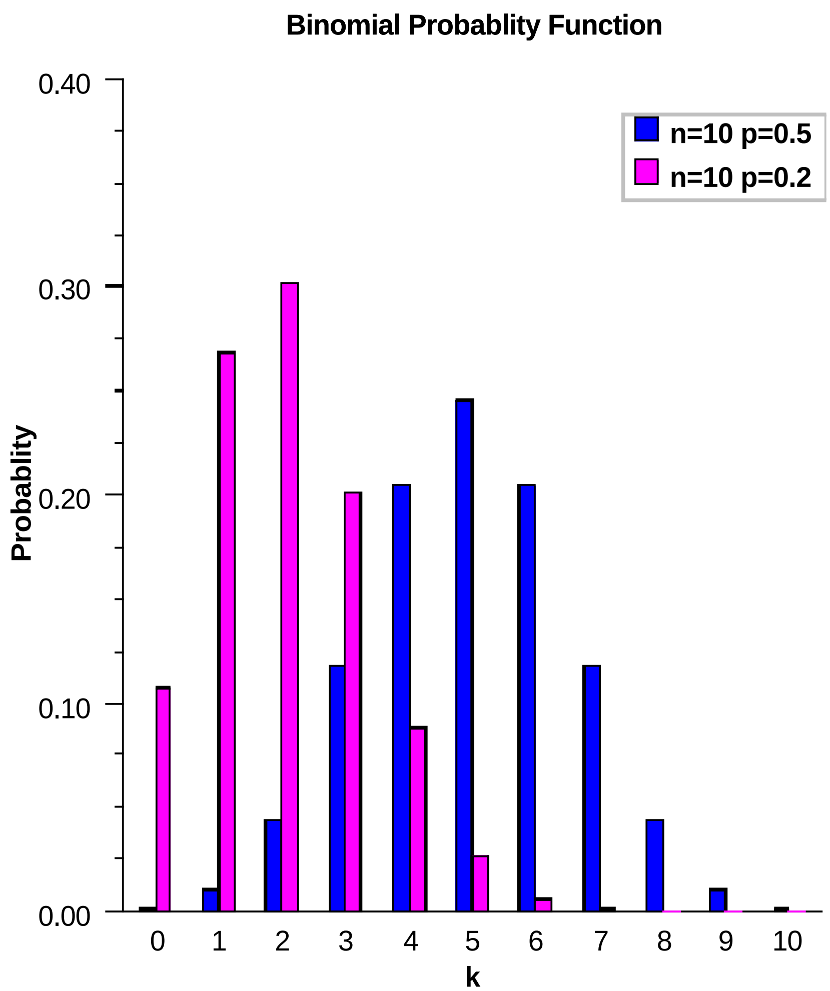
Evaluates the binomial probability density function.

Method `BinomialProb` evaluates the probability that a binomial random variable with parameters n and p 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, ε, 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 $p = 0$, `BinomialProb` is set to 0 if k is greater than 0 and to 1 otherwise; and for the case $p = 1$, `BinomialProb` is set to 0 if k is less than n and to 1 otherwise.



Parameters

`k` – An `int` specifying the argument for which the binomial distribution function is to be evaluated.

`n` – An `int` specifying the number of Bernoulli trials.

`p` – A `double` specifying the probability of success on each trial.

Returns

A `double` specifying the probability that a binomial random variable takes a value equal to k.

BivariateNormal

`static public double BivariateNormal(double x, double y, double rho)`

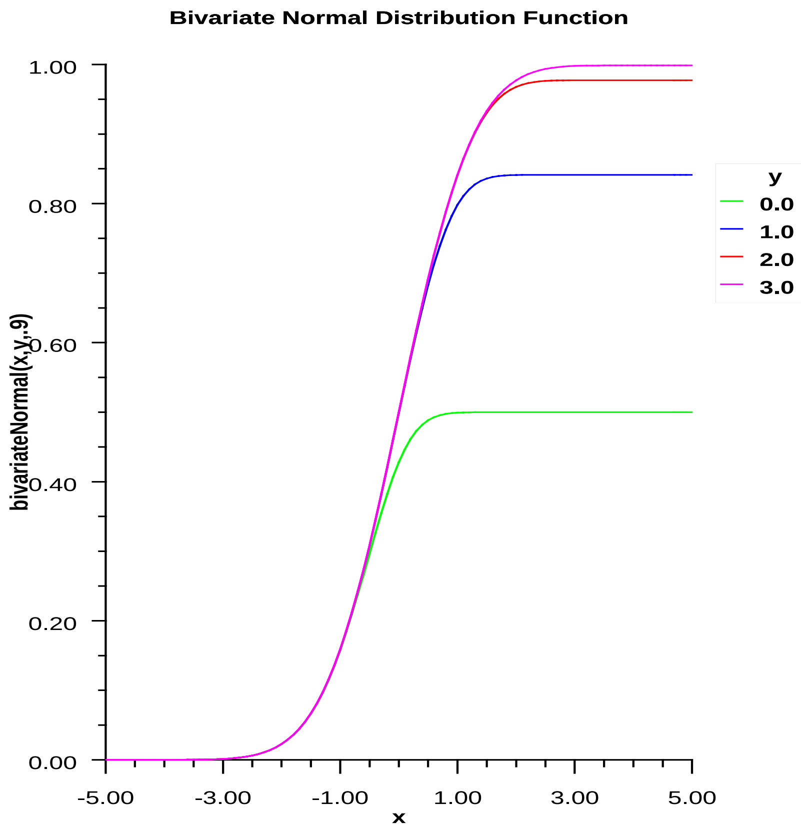
Description

Evaluates the bivariate normal cumulative probability distribution function.

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 $X \leq \texttt{x}$ and $Y \leq \texttt{y}$.



Parameters

`x` – is the x-coordinate of the point for which the bivariate normal distribution function is to be evaluated.

`y` – is the y-coordinate of the point for which the bivariate normal distribution function is to be evaluated.

`rho` – is 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`.

Chi

```
static public double Chi(double chsq, double df)
```

Description

Evaluates the chi-squared cumulative probability distribution function.

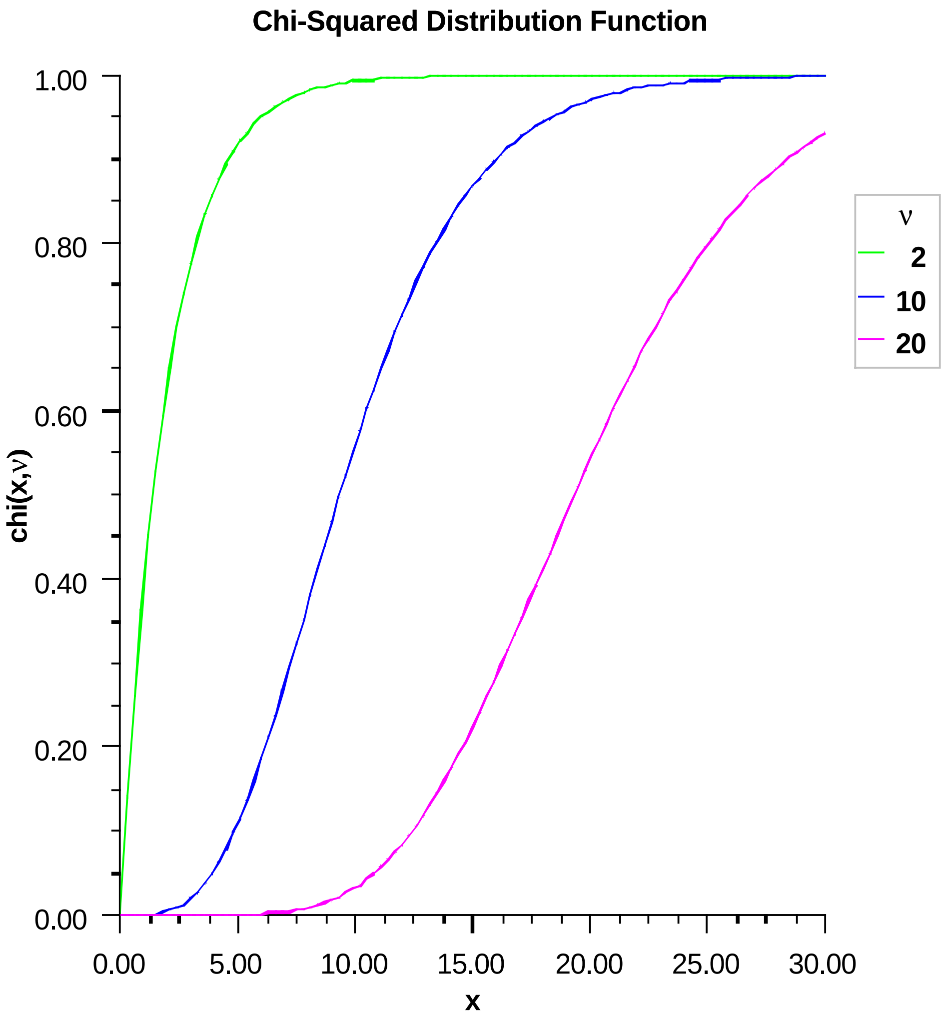
Method `Chi` evaluates the distribution function, F, of a chi-squared random variable with `df` degrees of freedom, that is, with $v = \text{df}$, and $x = \text{chsq}$,

$$F(x) = \frac{1}{2^{\nu/2}\Gamma(\nu/2)} \int_0^x e^{-t/2} t^{\nu/2-1} dt$$

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 $v > 65$, `Chi` uses the Wilson-Hilferty approximation (Abramowitz and Stegun 1964, equation 26.4.17) to the normal distribution, and method `Normal` is used to evaluate the normal distribution function.

For $v \leq 65$, `Chi` uses series expansions to evaluate the distribution function. If $x < \max(v/2, 26)$, `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.



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.

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.

ChiProb

`static public double ChiProb(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. This must be at least 0.5.

Returns

A `double` scalar value, the value of the probability density function at `chsq`.

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.

DiscreteUniform

`static 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. `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 less than or equal to `x`.

DiscreteUniformProb

```
static public double DiscreteUniformProb(int x, int n)
```

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

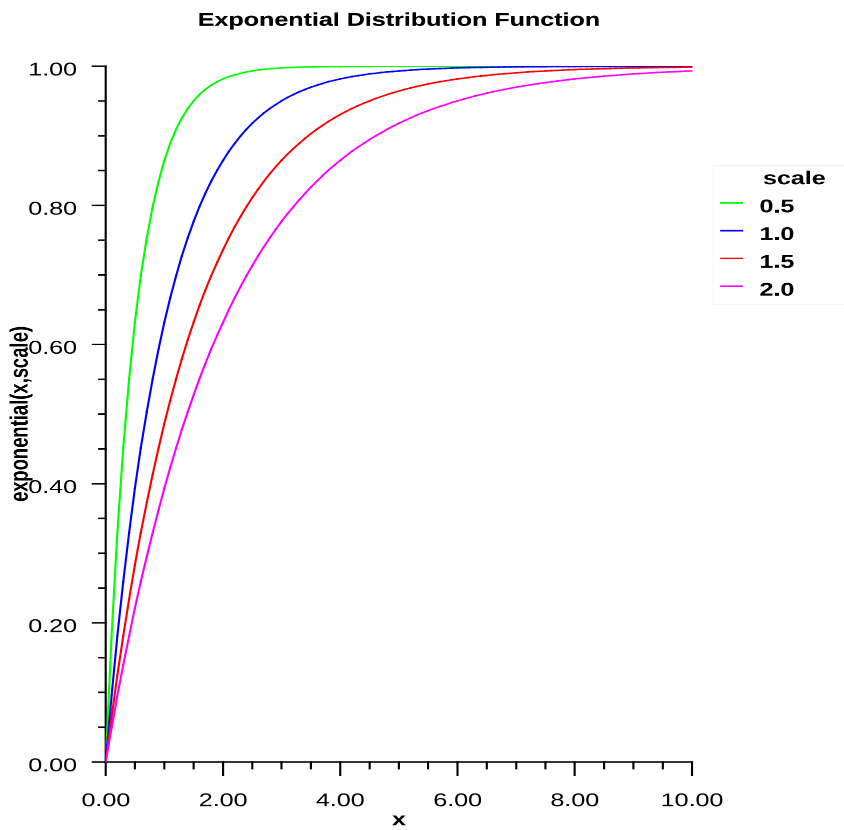
Evaluates the exponential cumulative probability distribution function.

Method `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,

$$F\left(x\right) = \frac{1}{\Gamma\left(a\right)}\int_0^x e^{-t/b}dt$$

where $\Gamma(\cdot)$ is the gamma function. (The gamma function is the integral from 0 to ∞ 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, `Gamma` uses a series expansion. Otherwise, a continued fraction expansion is used. (See Abramowitz and Stegun, 1964.)



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`.

ExponentialProb

```
static public double ExponentialProb(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

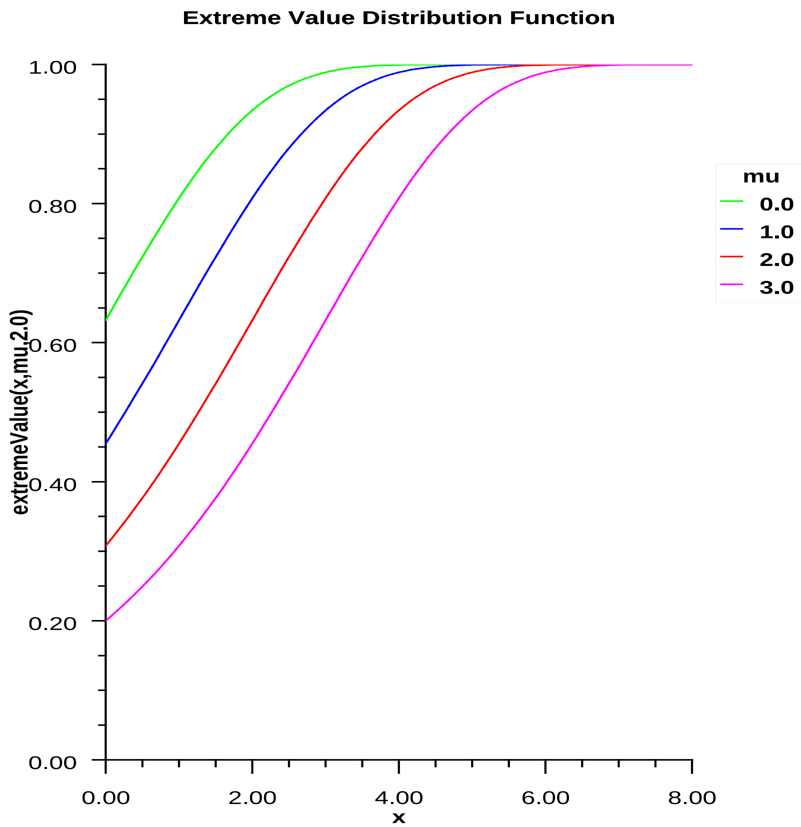
Evaluates the extreme value cumulative probability distribution function.

Method `ExtremeValue`, also known as the Gumbel minimum distribution, evaluates the extreme value distribution function, F, of a uniform random variable with location parameter μ and shape parameter β; that is,

$$F(x) = \int_0^x 1 - e^{-e^{\frac{x-\mu}{\beta}}} dt$$

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 $\frac{x_i - \mu}{\beta} = log(-log(1 - u_i))$.



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, μ.

beta – A double scalar value representing the scale parameter, β.

Returns

A `double` scalar value representing the probability that an extreme value random variable takes on a value less than or equal to `x`.

ExtremeValueProb

```
static public double ExtremeValueProb(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

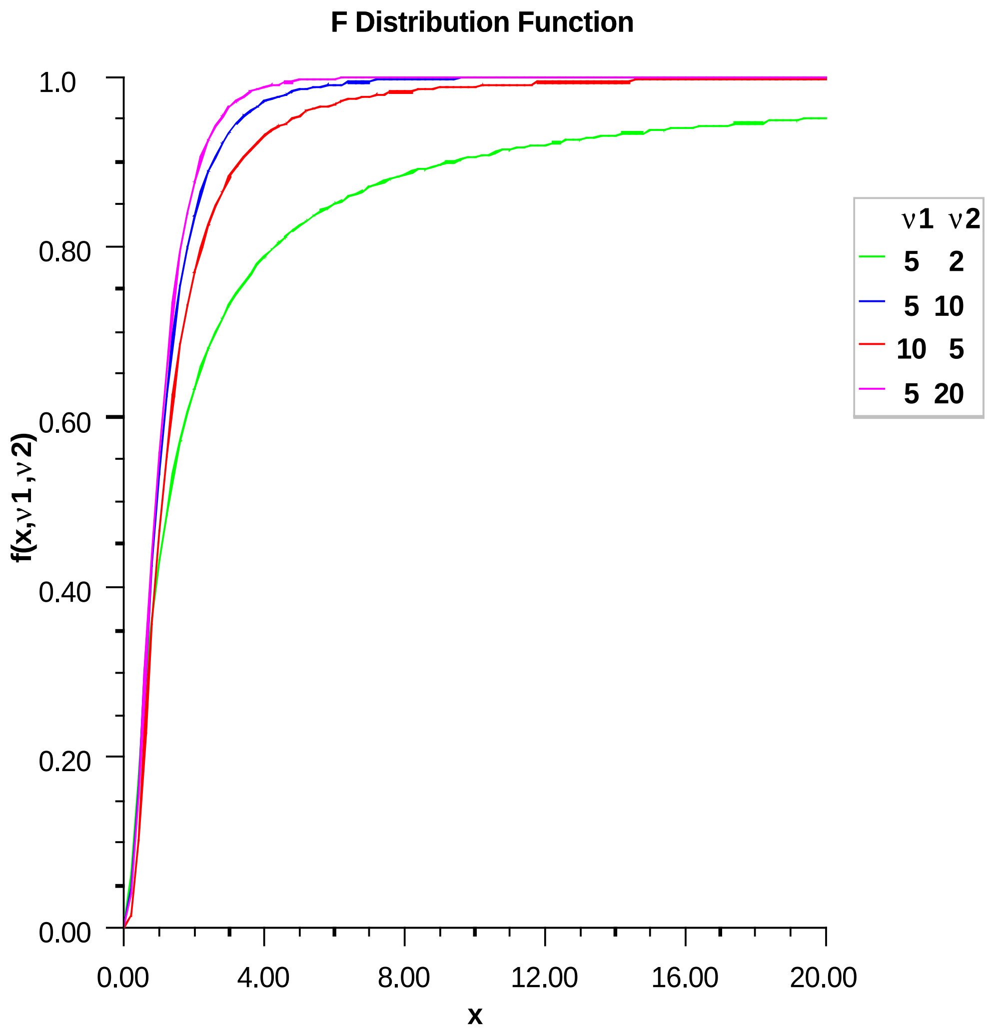
```
static public double F(double x, double dfn, double dfd)
```

Description

Evaluates the F cumulative probability distribution function.

`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 `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:

$$\mathrm{F}(X, dfn, dfd) = 1 - \mathrm{F}(1/X, dfd, dfn)$$



Parameters

`x` – A `double` specifying the argument at which the function is to be evaluated.

`dfn` – A `double` specifying the numerator degrees of freedom. It must be positive.

`dfd` – A `double` specifying the denominator degrees of freedom. It must be positive.

Returns

A `double` specifying the probability that an F random variable takes on a value less than or equal to x.

FProb

`static public double FProb(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`.

Gamma

`static public double Gamma(double x, double a)`

Description

Evaluates the gamma cumulative probability distribution function.

Method `Gamma` evaluates the distribution function, F, of a gamma random variable with shape parameter a; that is,

$$F(x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt$$

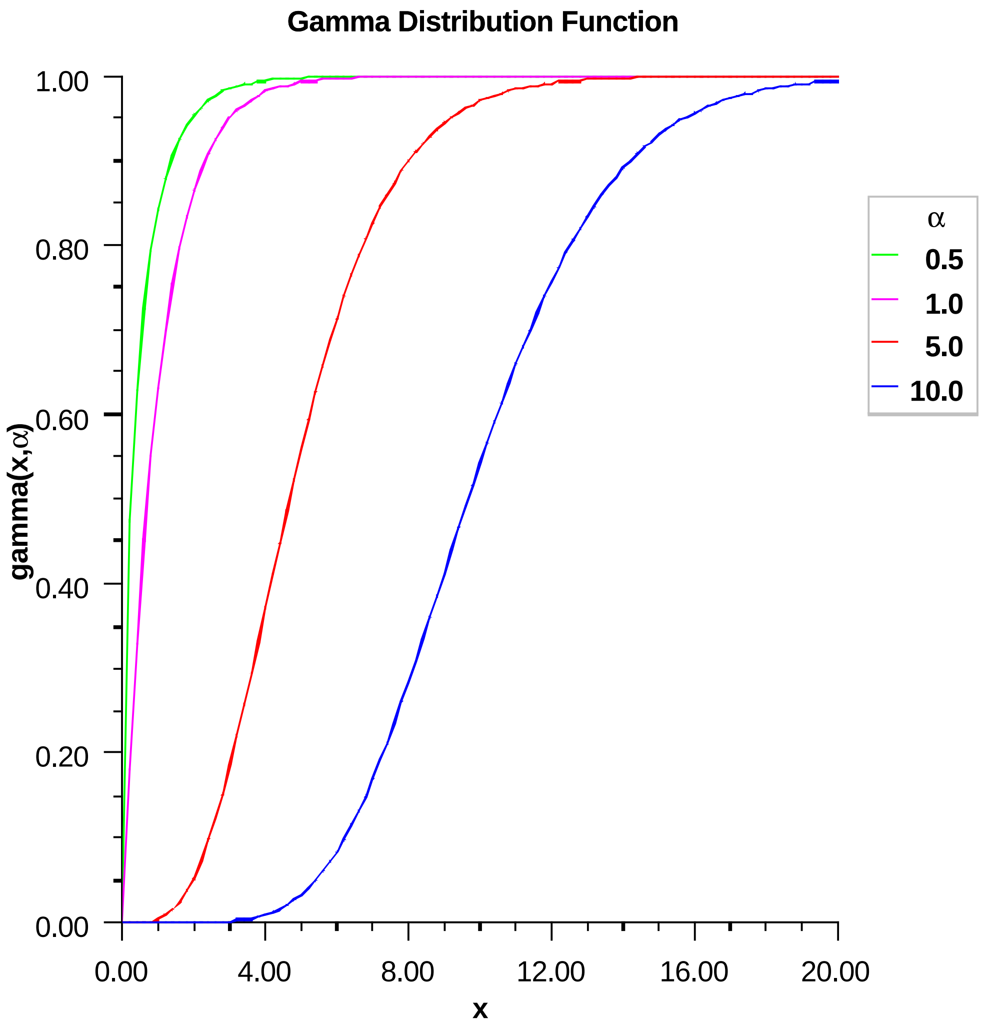
where $\Gamma(\cdot)$ is the gamma function. (The gamma function is the integral from 0 to ∞ of the same integrand as above). The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x.

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, ∞) is

$$f(t) = \frac{1}{b^a \Gamma(a)} e^{-(t-c)/b} (x-c)^{a-1}$$

If T is such a random variable with parameters a, b, and c, the probability that $T \leq t_0$ can be obtained from `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, `Gamma` uses a series expansion. Otherwise, a continued fraction expansion is used. (See Abramowitz and Stegun, 1964.)



Parameters

`x` – A `double` specifying the argument at which the function is to be evaluated.

`a` – A `double` specifying the shape parameter. This must be positive.

Returns

A `double` specifying the probability that a gamma random variable takes on a value less than or equal to x.

GammaProb

`static public double GammaProb(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.

Geometric

`static public double Geometric(int x, double p)`

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

p – An 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.

GeometricProb

`static public double GeometricProb(int x, double p)`

Description

Evaluates the discrete geometric probability density function.

Method GeometricProb evaluates the geometric distribution for the number of trials before the first success.

Parameters

x – An int argument for which the geometric probability function is to be evaluated.

p – 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`.

Hypergeometric

```
static public double Hypergeometric(int k, int sampleSize, int
  defectivesInLot, int lotSize)
```

Description

Evaluates the hypergeometric cumulative probability distribution function.

Method `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

$$\Pr\left(X=j\right)=\frac{\binom{m}{j}\binom{l-m}{n-j}}{\binom{l}{n}}\text{for } j=i,\ i+1,\ i+2,\ \ldots,\ \min\left(n,m\right)$$

where $i=\max(0,n-l+m)$.

If k is greater than or equal to i and less than or equal to $\min(n,m)$, `Hypergeometric` sums the terms in this expression for j going from i up to k. Otherwise, 0 or 1 is returned, as appropriate. So, as to avoid rounding in the accumulation, `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 $(m+1)(n+1)/(l+2)$.

Parameters

`k` – An `int` specifying the argument at which the function is to be evaluated.

`sampleSize` – An `int` specifying the sample size, n.

`defectivesInLot` – An `int` specifying the number of defectives in the lot, m.

`lotSize` – An `int` specifying the lot size, l.

Returns

A `double` specifying the probability that a hypergeometric random variable takes a value less than or equal to k.

HypergeometricProb

```
static public double HypergeometricProb(int k, int sampleSize, int
  defectivesInLot, int lotSize)
```

Description

Evaluates the hypergeometric probability density function.

Method `HypergeometricProb` evaluates the probability 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:

$$\Pr\left(X=k\right)=\frac{\binom{m}{k}\binom{l-m}{n-k}}{\binom{l}{n}}\text{for } \ k=i,\ i+1,\, i+2\ \ldots,\ \min\left(n,m\right)$$

where $i = max(0,\ n - l + m)$. `HypergeometricProb` evaluates the expression using log gamma functions.

Parameters

`k` – An `int` specifying the argument at which the function is to be evaluated.

`sampleSize` – An `int` specifying the sample size, n.

`defectivesInLot` – An `int` specifying the number of defectives in the lot, m.

`lotSize` – An `int` specifying the lot size, l.

Returns

A `double` specifying the probability that a hypergeometric random variable takes a value equal to k.

InverseBeta

`static public double InverseBeta(double p, double pin, double qin)`

Description

Evaluates the inverse of the beta cumulative probability distribution function.

Method `InverseBeta` evaluates the inverse distribution function of a beta random variable with parameters `pin` and `qin`, that is, with $P = p$, $p = pin$, and $q = qin$, it determines x (equal to `InverseBeta (p, pin, qin)`), such that

$$P=\frac{\Gamma\left(p\right)\Gamma\left(q\right)}{\Gamma\left(p+q\right)}\int_0^x t^{p-1}\left(1-t\right)^{q-1}dt$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to x is P.

Parameters

`p` – A `double` specifying the probability for which the inverse of the beta CDF is to be evaluated.

`pin` – A `double` specifying the first beta distribution parameter.

`qin` – A `double` specifying the second beta distribution parameter.

Returns

A `double` specifying the probability that a beta random variable takes a value less than or equal to this value is p.

InverseChi

`static public double InverseChi(double p, double df)`

Description

Evaluates the inverse of the chi-squared cumulative probability distribution function.

Method `InverseChi` evaluates the inverse distribution function of a chi-squared random variable with `df` degrees of freedom, that is, with $P = p$ and $v = df$, it determines x (equal to `InverseChi(p, df)`), such that

$$P = \frac{1}{2^{\nu/2}\Gamma\left(\nu/2\right)}\int_0^x e^{-t/2}t^{\nu/2-1}dt$$

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 $v < 40$, `InverseChi` 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 `Chi`.

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 `InverseNormal` 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.

Parameters

`p` – A `double` specifying the probability for which the inverse chi-squared 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 value less than or equal to this value is p.

InverseDiscreteUniform

`static public int InverseDiscreteUniform(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`.

InverseExponential

`static public double InverseExponential(double p, double scale)`

Description

Evaluates the inverse of the exponential cumulative probability distribution function.

Method `InverseExponential` evaluates the inverse distribution function of a gamma random variable with scale parameter $=b$ and shape parameter a=1.0, that is, it determines $x = \text{inverseExponential}(p, 1.0))$, such that

$$P = \frac{1}{\Gamma(a)} \int_o^x e^{-t/b} dt$$

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 documentation for routine `Gamma` for further discussion of the gamma distribution.

`InverseExponential` uses bisection and modified regula falsi to invert the distribution function, which is evaluated using method `Gamma`.

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, b.

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`.

InverseExtremeValue

`static public double InverseExtremeValue(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`.

InverseF

`static public double InverseF(double p, double dfn, double dfd)`

Description

Returns the inverse of the F cumulative probability distribution function.

Method `InverseF` 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 `InverseBeta`. 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 $q = v_2/2$. If $P \leq 0.5$, `InverseF` 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:

$$\mathrm{F}(X, dfn, dfd) = 1 - \mathrm{F}(1/X, dfd, dfn)$$

Parameters

`p` – A `double` specifying 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` specifying the numerator degrees of freedom. It must be positive.

`dfd` – A `double` specifying the denominator degrees of freedom. It must be positive.

Returns

A `double` specifying the probability that an F random variable takes a value less than or equal to this value is p.

InverseGamma

`static public double InverseGamma(double p, double a)`

Description

Evaluates the inverse of the gamma cumulative probability distribution function.

Method `InverseGamma` evaluates the inverse distribution function of a gamma random variable with shape parameter a, that is, it determines $x = \mathrm{InverseGamma}(p, a)$, such that

$$P = \frac{1}{\Gamma(a)} \int_o^x e^{-t} t^{a-1} dt$$

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 documentation for routine `Gamma` for further discussion of the gamma distribution.

`InverseGamma` uses bisection and modified regula falsi to invert the distribution function, which is evaluated using method `Gamma`.

Parameters

`p` – A `double` specifying the probability at which the function is to be evaluated.

`a` – A `double` specifying the shape parameter, a. This must be positive.

Returns

A `double` specifying the probability that a gamma random variable takes a value less than or equal to this value is p.

InverseGeometric

```
static public double InverseGeometric(double r, double p)
```

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.

`p` – 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 `r`.

InverseLogNormal

```
static public double InverseLogNormal(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`.

InverseNoncentralchi

`static public double InverseNoncentralchi(double p, double df, double alam)`

Description

Evaluates the inverse of the noncentral chi-squared cumulative probability distribution function.

Method `InverseNoncentralchi` evaluates the inverse distribution function of a noncentral chi-squared random variable with `df` degrees of freedom and noncentrality parameter `alam`, that is, with $P = p$, $\nu = \mathrm{df}$, and $\lambda = \mathrm{alam}$, it determines $c_0 =$ `InverseNoncentralchi(p, df, alam)`), such that

$$P = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} \left(\lambda/2\right)^i}{i!} \int_0^{c_0} \frac{x^{(\nu+2i)/2-1} e^{-x/2}}{2^{(\nu+2i)/2} \Gamma\left(\frac{\nu+2i}{2}\right)} dx$$

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 `InverseNoncentralchi` uses bisection and modified regula falsi to invert the distribution function, which is evaluated using `noncentralchi`. See `Noncentralchi` for an alternative definition of the noncentral chi-squared random variable in terms of normal random variables.

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`.

InverseNoncentralstudentsT

```
static public double InverseNoncentralstudentsT(double p, int idf, double
  delta)
```

Description

Evaluates the inverse of the noncentral Student's t cumulative probability distribution function.

Method `InverseNoncentralstudentsT` 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 $t_0 =$ `InverseNoncentralstudentsT(p, idf, delta)`, such that

$$P = \int_{-\infty}^{t_0} \frac{\nu^{\nu/2} e^{-\delta^2/2}}{\sqrt{\pi}\Gamma\left(\nu/2\right)\left(\nu + x^2\right)^{(\nu+1)/2}} \sum_{i=0}^{\infty} \Gamma\left(\left(\nu + i + 1\right)/2\right)\left(\frac{\delta^i}{i!}\right)\left(\frac{2x^2}{\nu + x^2}\right)^{i/2} dx$$

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 `NoncentralstudentsT` for an alternative definition in terms of normal and chi-squared random variables. The method `InverseNoncentralstudentsT` uses bisection and modified regula falsi to invert the distribution function, which is evaluated using `NoncentralstudentsT`.

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`.

InverseNormal

```
static public double InverseNormal(double p)
```

Description

Evaluates the inverse of the normal (Gaussian) cumulative probability distribution function.

Method `InverseNormal` evaluates the inverse of the distribution function, Φ, of a standard normal (Gaussian) random variable, that is, `InverseNormal` (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.

Parameter

`p` – A `double` specifying the probability at which the function is to be evaluated.

Returns

A `double` specifying the probability that a standard normal random variable takes a value less than or equal to this value is p.

InverseRayleigh

`static public double InverseRayleigh(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`.

InverseStudentsT

`static public double InverseStudentsT(double p, double df)`

Description

Returns the inverse of the Student's t cumulative probability distribution function.

`InverseStudentsT` 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 `InverseBeta` 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.

Parameters

`p` – A `double` specifying the probability for which the inverse Student's t function is to be evaluated.

`df` – A `double` specifying the number of degrees of freedom. This must be at least one.

Returns

A `double` specifying the probability that a Student's t random variable takes a value less than or equal to this value is p.

InverseUniform

`static public double InverseUniform(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`.

InverseWeibull

`static public double InverseWeibull(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`.

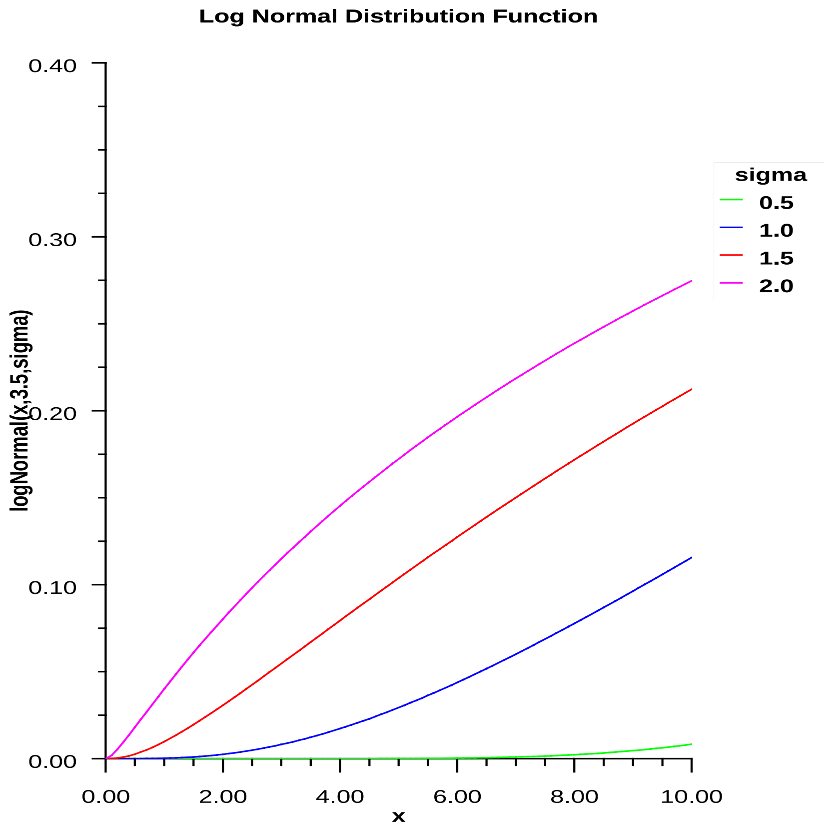
LogNormal

`static public double LogNormal(double x, double mu, double sigma)`

Description

Evaluates the standard lognormal cumulative probability distribution function.

$$F(x) = \frac{1}{x^{\sigma}\sqrt{2\pi}} \int \frac{1}{t} e^{-\frac{\ln t - \mu^2}{2\sigma^2}}$$



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`.

LogNormalProb

`static public double LogNormalProb(double x, double mu, double sigma)`

Description

Evaluates the standard lognormal probability density function.

$$F(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}$$

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`.

Noncentralchi

`static public double Noncentralchi(double chsq, double df, double alam)`

Description

Evaluates the noncentral chi-squared cumulative probability distribution function.

Method `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 $\nu = \mathrm{df}$, $\lambda = \mathrm{alam}$, and $x = \mathrm{chsq}$,

$$F(x) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} \int_0^x \frac{t^{(\nu+2i)/2-1} e^{-t/2}}{2^{(\nu+2i)/2} \Gamma\left(\frac{\nu+2i}{2}\right)} dt$$

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 μ_i and variances equal to one and

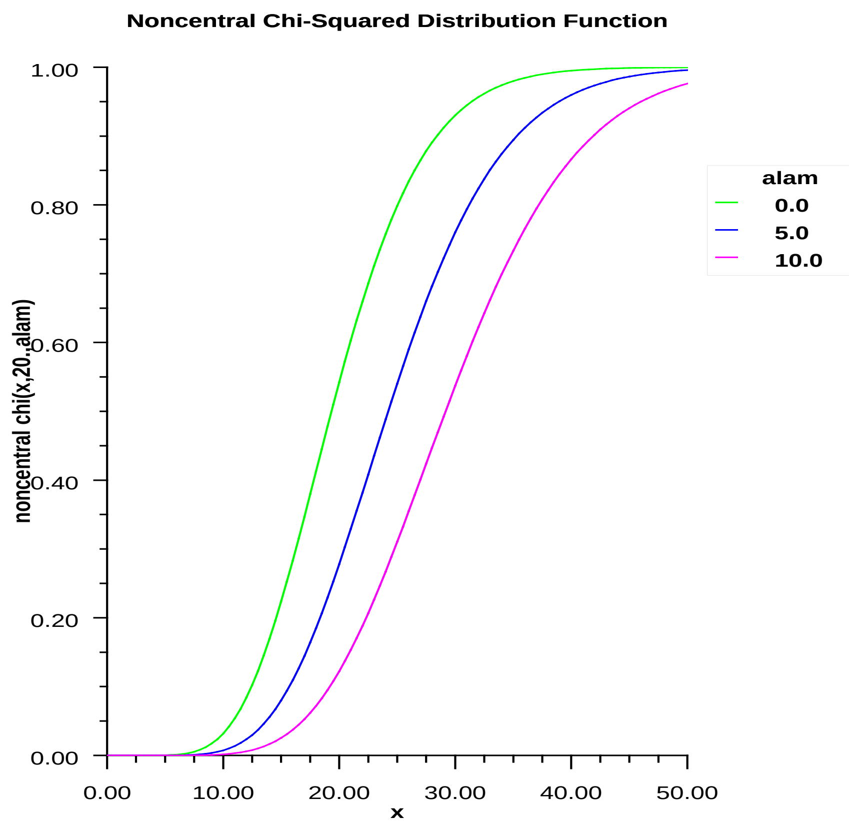
$$X = \sum_{i=1}^{n} {Y_i}^2$$

then X 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.

`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.



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. This must be at least 0.5.

`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 representing the probability that a chi-squared random variable takes a value less than or equal to `chsq`.

NoncentralstudentsT

```
static public double NoncentralstudentsT(double t, int idf, double delta)
```

Description

Evaluates the noncentral Student's t cumulative probability distribution function.

Method `NoncentralstudentsT` evaluates the distribution function `F` of a noncentral t random variable with `idf` degrees of freedom and noncentrality parameter `delta`; that is, with $\nu = idf$, $\delta = delta$, and $t_0 = t$,

$$F(t_0) = \int_{-\infty}^{t_0} \frac{\nu^{\nu/2} e^{-\delta^2/2}}{\sqrt{\pi}\Gamma\left(\nu/2\right)\left(\nu + x^2\right)^{(\nu+1)/2}} \sum_{i=0}^{\infty} \Gamma\left(\left(\nu + i + 1\right)/2\right)\left(\frac{\delta^i}{i!}\right)\left(\frac{2x^2}{\nu + x^2}\right)^{i/2} dx$$

where $\Gamma(\cdot)$ 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 δ and variance equal to one, u has an independent chi-squared distribution with ν degrees of freedom, and

$$x = w/\sqrt{u/\nu}$$

then x has a noncentral t distribution with ν degrees of freedom and noncentrality parameter δ.

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 `NoncentralstudentsT` uses the method of Owen (1962, 1965), which uses repeated integration by parts on that alternate expression for the 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`.

Normal

`static public double Normal(double x)`

Description

Evaluates the normal (Gaussian) cumulative probability distribution function.

Method `Normal` evaluates the distribution function, Φ, of a standard normal (Gaussian) random variable, that is,

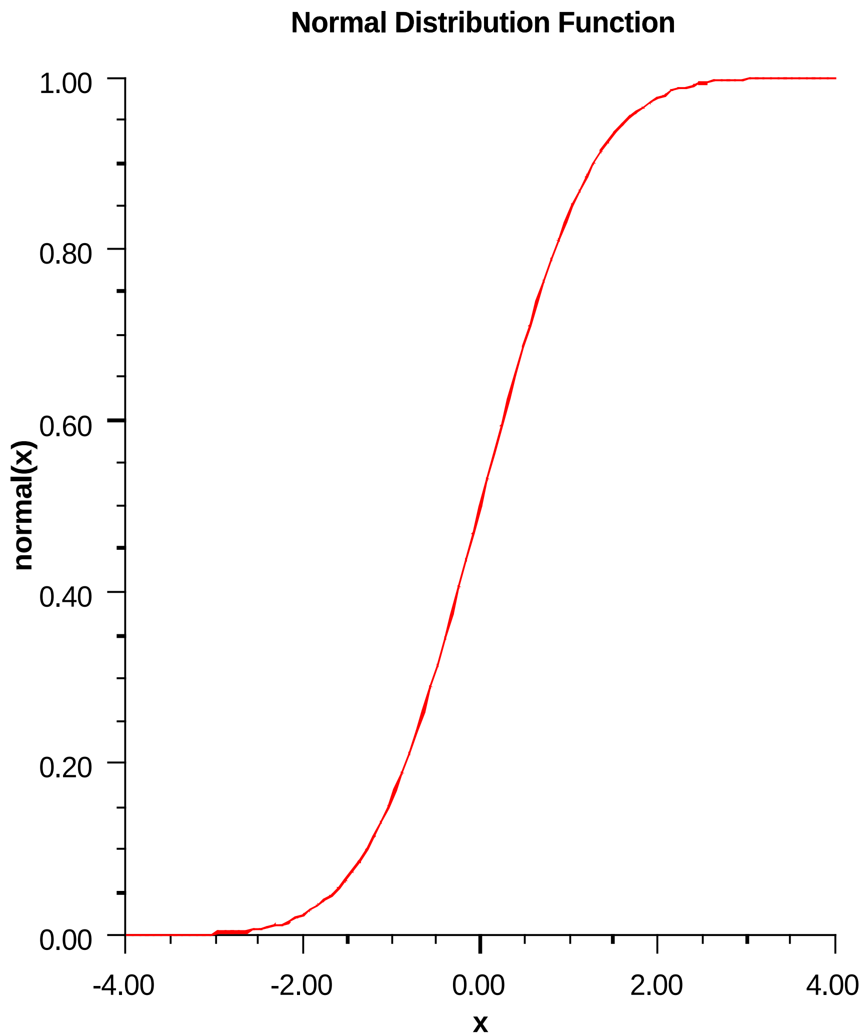
$$\Phi\left(x\right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt$$

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 `Normal` is the distribution function) has mean of 0 and variance of 1. The probability that a normal random variable with mean μ and variance σ^2 is less than y s given by `Normal` evaluated at $(y - \mu)/\sigma$.

$\Phi(x)$ is evaluated by use of the complementary error function, erfc. The relationship is:

$$\Phi(x) = \mathrm{erfc}(-x/\sqrt{2.0})/2$$



Parameter

x – A double specifying the argument at which the function is to be evaluated.

Returns

A double specifying the probability that a normal variable takes a value less than or equal to x.

Poisson

`static public double Poisson(int k, double theta)`

Description

Evaluates the Poisson cumulative probability distribution function.

`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 $\theta = theta$) is

$$f(x) = e^{-\theta}\theta^x/x! \;\; for\, x = 0,\, 1,\, 2,\, \ldots$$

The individual terms are calculated from the tails of the distribution to the mode of the distribution and summed. `Poisson` uses the recursive relationship

$$f\left(x+1\right) = f\left(x\right)\left(\theta/\left(x+1\right)\right), \;\; for\, x = 0,\, 1,\, 2,\, \ldots k-1$$

with $f(0) = e^{-\theta}$.

Parameters

`k` – An `int` specifying the argument for which the Poisson distribution function is to be evaluated.

`theta` – A `double` specifying the mean of the Poisson distribution.

Returns

A `double` specifying the probability that a Poisson random variable takes a value less than or equal to k.

PoissonProb

`static public double PoissonProb(int k, double theta)`

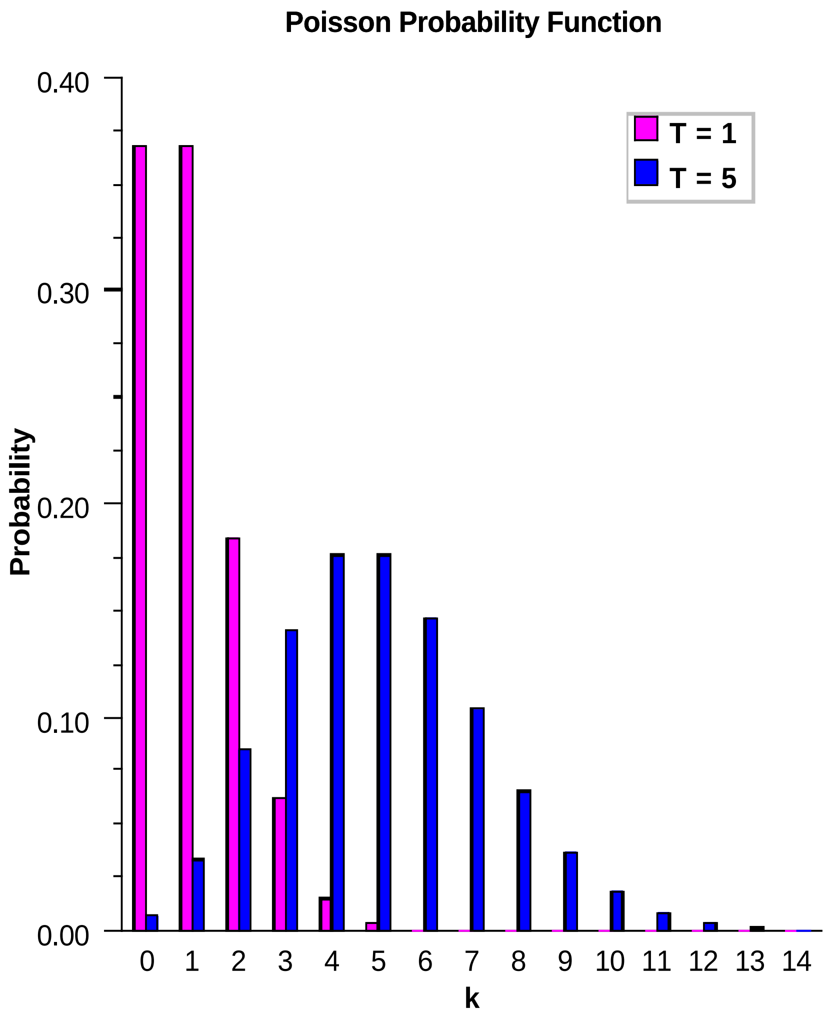
Description

Evaluates the Poisson probability density function.

Method `PoissonProb` evaluates the probability 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

$$f(x) = e^{-\theta}\;\theta^k/k!, \quad for\, k = 0,\; 1,\; 2,\; \ldots$$

`PoissonProb` evaluates this function directly, taking logarithms and using the log gamma function.



Parameters

`k` – An `int` specifying the argument for which the Poisson probability function is to be evaluated.

`theta` – A `double` specifying the mean of the Poisson distribution.

Returns

A `double` specifying the probability that a Poisson random variable takes a value equal to k.

Rayleigh

`static public double Rayleigh(double x, double alpha)`

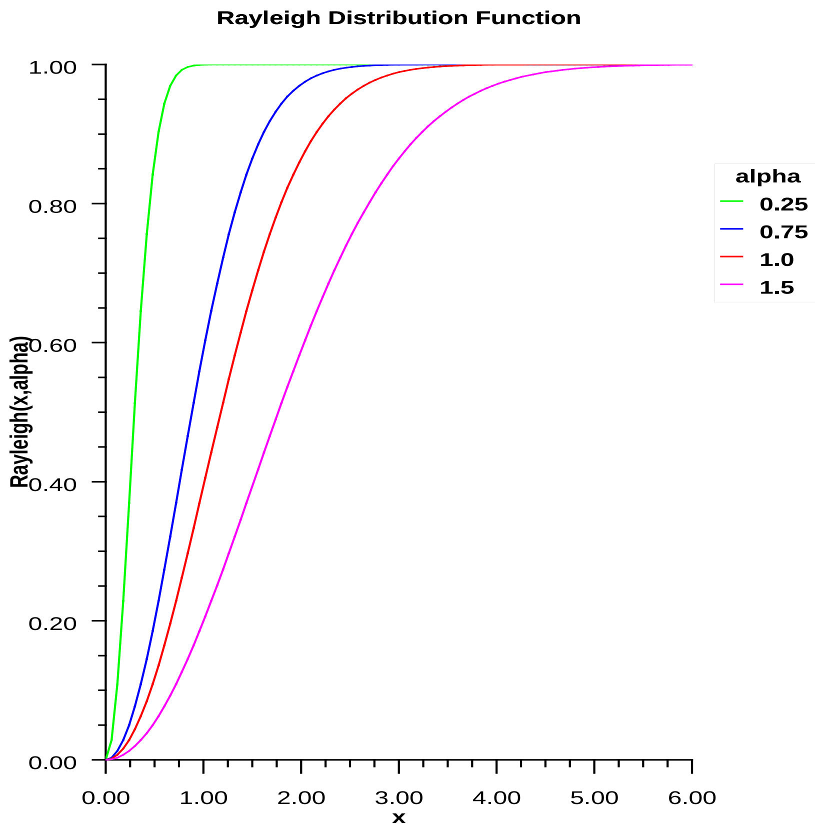
Description

Evaluates the Rayleigh cumulative probability distribution function.

Method `Rayleigh` is a special case of Weibull distribution function where the shape parameter `gamma` is 2.0; that is,

$F(x) = 1 - e^{-\frac{x^2}{2\alpha^2}}$

where `alpha` is the scale parameter.



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`.

RayleighProb

`static public double RayleighProb(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`.

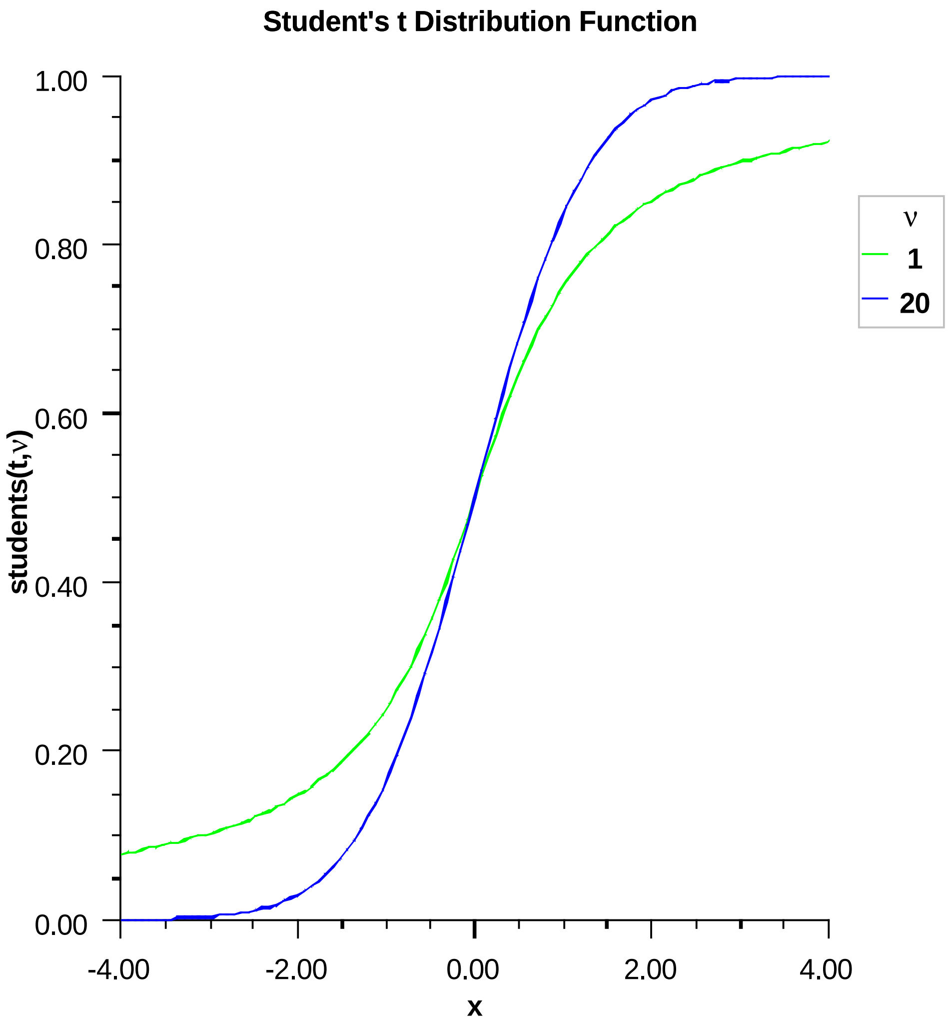
StudentsT

`static public double StudentsT(double t, double df)`

Description

Evaluates the Student's t cumulative probability distribution function.

Method `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 `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.



Parameters

`t` – 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 one.

Returns

A `double` specifying the probability that a Student's t random variable takes a value less than or equal to t.

Uniform

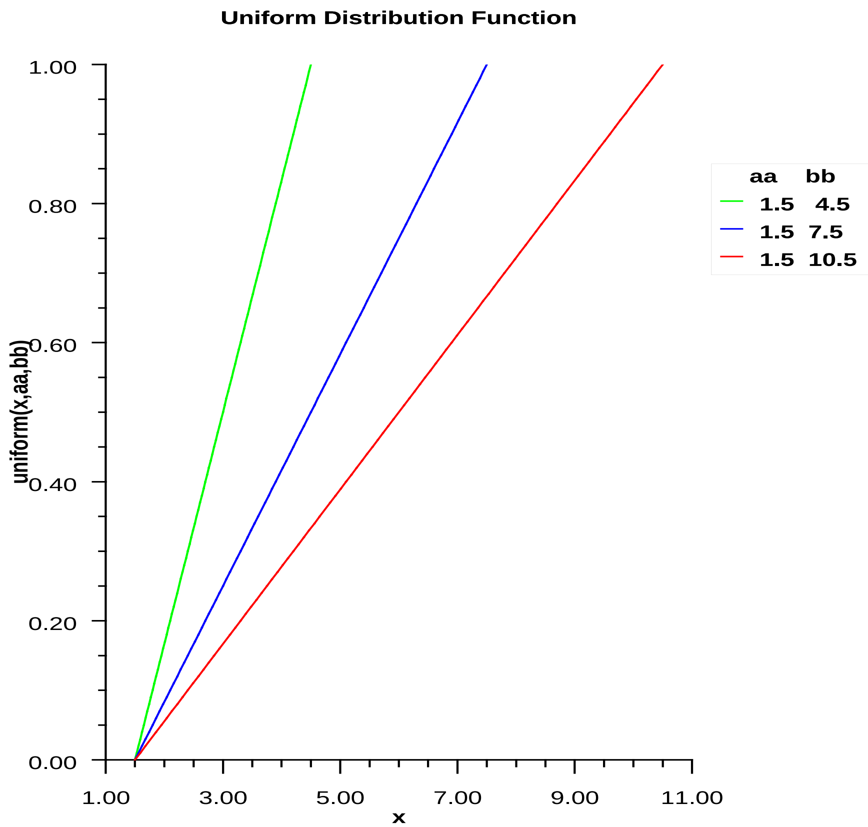
`static public double Uniform(double x, double aa, double bb)`

Description

Evaluates the uniform cumulative probability distribution function.

Method `Uniform` evaluates the distribution function, F, of a uniform random variable with location parameter aa and scale parameter bb; that is,

$$\mathrm{f(x)}=\begin{cases} 0, & \text{if } x < aa \\ \frac{x-aa}{bb-aa}, & \text{if } aa \le x \le bb \\ 1, & \text{if } x > bb \end{cases}$$



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`.

Weibull

`static public double Weibull(double x, double gamma, double alpha)`

Description

Evaluates the Weibull cumulative probability distribution function.

Parameters

`x` – A `double` specifying the argument at which the function is to be evaluated. It must be non-negative.

`gamma` – A `double` specifying the shape parameter.

`alpha` – A `double` specifying the scale parameter.

Returns

A `double` specifying the probability that a Weibull random variable takes a value less than or equal to x.

WeibullProb

`static public double WeibullProb(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 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.

```
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("beta(.5, 12, 12) is " + result);

        // Inverse Beta
        x = .5;
        p = 12;
        q = 12;
        result = Cdf.InverseBeta(x, p, q);
        Console.Out.WriteLine("inversebeta(.5, 12, 12) is " + result);

        // binomial
        k = 3;
        n = 5;
        prob = .95;
        result = Cdf.Binomial(k, n, prob);
        Console.Out.WriteLine("binomial(3, 5, .95) is " + result);

        // Chi
        x = .15;
        n = 2;
        result = Cdf.Chi(x, n);
        Console.Out.WriteLine("chi(.15, 2) is " + result);

        // Inverse Chi
        prob = .99;
        n = 2;
        result = Cdf.InverseChi(prob, n);
        Console.Out.WriteLine("inverseChi(.99, 2) is " + result);
    }
}
```

Output

```
beta(.5, 12, 12) is 0.500000000000002
inversebeta(.5, 12, 12) is 0.5
binomial(3, 5, .95) is 0.0225925
chi(.15, 2) is 0.0722565136714471
inverseChi(.99, 2) is 9.21034037197624
```

ICdfFunction Interface

Summary

Interface for the user-supplied cumulative distribution function to be used by `InverseCdf` and `ChiSquaredTest`.

`public interface Imsl.Stat.ICdfFunction`

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

Summary

Inverse of user-supplied cumulative distribution function.

`public class Imsl.Stat.InverseCdf`

Property

Tolerance

`public double Tolerance {get; set; }`

Description

The tolerance to be used as the convergence criterion.

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 cummulative distribution function.

The cdf function must be continuous and strictly monotone.

Parameter

`cdf` – A `ICdfFunction` object that contains the user-supplied function to be inverted.

Method

Eval

`public double Eval(double p, double guess)`

Description

Evaluates the inverse CDF function.

Cdf(`InverseCdf`) is ”close” to `p`.

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`.

`Imsl.Stat.DidNotConvergeException` id is thrown if the interation to find the inverse of the CDF did not converge.

Description

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.

Example: Inverse of a User-Supplied Cumulative Distribution Function

In this example, 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.

```
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
```

Chapter 21: Random Number Generation

Types

Random Class

Summary

Generate uniform and non-uniform random number distributions.

```
public class Imsl.Stat.Random :  Random
```

Property

Multiplier

```
public System.Int64 Multiplier {get; set; }
```

Description

The multiplier for a linear congruential random number generator.

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.

Constructors

Random

`public Random()`

Description

Constructor for the Random number generator class.

Random

`public Random(int seed)`

Description

Constructor for the Random number generator class with supplied seed.

Parameter

`seed` – A `int` which represents the random number generator seed.

Random

`public Random(Imsl.Stat.Random.BaseGenerator baseGenerator)`

Description

Constructor for the Random number generator class with an alternate basic number genrator.

Parameter

`baseGenerator` – A `BaseGenerator` used to override the method `next`.

Methods

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.

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

$$f(x) = \frac{\Gamma(p+q)}{\Gamma(p)\Gamma(q)} x^{p-1} (1-x)^{q-1} \quad for\, 0 \le x \le 1$$

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 ε, where ε is the smallest positive number such that $1.0 - \varepsilon$ is less than 1.0.

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.

NextBinomial

```
virtual public int NextBinomial(int n, double p)
```

Description

Generate a pseudorandom number from a Binomial distribution.

`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.)

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.

NextCauchy

```
virtual public double NextCauchy()
```

Description

Generates a pseudorandom number from a Cauchy distribution.

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

$$f(x) = \frac{s}{\pi\left[s^2 + (x-t)^2\right]}$$

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.

Returns

A `double` which specifies a pseudorandom number from a Cauchy distribution.

NextChiSquared

`virtual public double NextChiSquared(double df)`

Description

Generates a pseudorandom number from a Chi-squared distribution.

`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 $n = \mathrm{df}/2$ 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.

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.

NextDouble

`override public double NextDouble()`

Description

Generates the next pseudorandom number.

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.

Returns

A `double` which specifies the next pseudorandom value from this random number generator's sequence.

NextExponential

`virtual public double NextExponential()`

Description

Generates a pseudorandom number from a standard exponential distribution.

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 `THETA` can be generated by using `NextExponential` and then multiplying the result by `THETA`.

Returns

A `double` which specifies a pseudorandom number from a standard exponential distribution.

NextExponentialMix

```
virtual public double NextExponentialMix(double theta1, double theta2,
  double p)
```

Description

Generate a pseudorandom number from a mixture of two exponential distributions.

The probability density function is

$$f(x) = \frac{p}{\theta} e^{-x/\theta_1} + \frac{1-p}{\theta_2} e^{-x/\theta_2} \quad for\, x > 0$$

where $p = $ p, $\theta_1 = theta1$, and $\theta_2 = theta2$.

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 θ_1, and with probability *1 - p* generates an exponential with mean θ_2. When p is greater than 1, but less than $\theta_1/(\theta_1 - \theta_2)$, then either an exponential deviate with mean θ_2 or the sum of two exponentials with means θ_1 and θ_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.

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.

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.

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.

Returns

A `float` which specifies the next pseudorandom value from this random number generator's sequence.

NextGamma

`virtual public double NextGamma(double a)`

Description

Generates a pseudorandom number from a standard gamma distribution.

Method `NextGamma` generates pseudorandom numbers from a gamma distribution with shape parameter a. The probability density function is

$$P = \frac{1}{\Gamma(a)} \int_o^x e^{-t} t^{a-1} dt$$

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.

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.

NextGeometric

`virtual public int NextGeometric(double p)`

Description

Generate a pseudorandom number from a geometric distribution.

`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.

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.

NextHypergeometric

```
virtual public int NextHypergeometric(int n, int m, int l)
```

Description

Generate a pseudorandom number from a hypergeometric distribution.

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.

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.

NextLogarithmic

`virtual public int NextLogarithmic(double a)`

Description

Generate a pseudorandom number from a logarithmic distribution.

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.

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.

NextLogNormal

```
virtual public double NextLogNormal(double mean, double stdev)
```

Description

Generate a pseudorandom number from a lognormal distribution.

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 $\mu = mean$ and $\sigma = stdev$, the probability density function for the lognormal distribution is

$$f\left(x\right)=\frac{1}{\sigma x\sqrt{2\pi}}\exp\left[-\frac{1}{2\sigma^2}\left(\ln x-\mu\right)^2\right]\ for\, x>0$$

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.

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.

NextMultivariateNormal

```
virtual public double[] NextMultivariateNormal(int k, Imsl.Math.Cholesky
  matrix)
```

Description

Generate pseudorandom numbers from a multivariate normal distribution.

`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.

Parameters

`k` – A `int` which specifies the length of the multivariate normal vectors.

`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.

NextNegativeBinomial

```
virtual public int NextNegativeBinomial(double rk, double p)
```

Description

Generate a pseudorandom number from a negative Binomial distribution.

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.

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.

NextNormal

```
virtual public double NextNormal()
```

Description

Generate a pseudorandom number from a standard normal distribution using an inverse CDF method.

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.

Returns

A `double` which represents a pseudorandom number from a standard normal distribution.

NextNormalAR

```
virtual public double NextNormalAR()
```

Description

Generate a pseudorandom number from a standard normal distribution using an acceptance/rejection method.

`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.

Returns

A `double` which represents a pseudorandom number from a standard normal distribution.

NextPoisson

```
virtual public int NextPoisson(double theta)
```

Description

Generate a pseudorandom number from a Poisson distribution.

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 $\theta = \text{theta}$) is

$$f(x) = e^{-\theta}\,\theta^x/x!$$

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.

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.

NextRayleigh

```
virtual public double NextRayleigh(double alpha)
```

Description

Generate a pseudorandom number from a Rayleigh distribution.

Method `nextRayleigh` generates pseudorandom numbers from a Rayleigh distribution with scale parameter *alpha*.

Parameter

`alpha` – A `double` which specifies the scale parameter of the Rayleigh distribution

Returns

A `double`, a pseudorandom number from a Rayleigh distribution

NextStudentsT

```
virtual public double NextStudentsT(double df)
```

Description

Generate a pseudorandom number from a Student's t distribution.

`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`.

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.

NextTriangular

`virtual public double NextTriangular()`

Description

Generate a pseudorandom number from a triangular distribution on the interval (0,1).

The probability density function is $f(x) = 4x$, for $0 \le x \le .5$, and $f(x) = 4(1 - x)$, for $.5 < x \le 1$. `NextTriangular` uses an inverse CDF technique.

Returns

A `double` which specifies a pseudorandom number from a triangular distribution on the interval (0,1).

NextVonMises

`virtual public double NextVonMises(double c)`

Description

Generate a pseudorandom number from a von Mises distribution.

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

$$f\left(x\right) = \frac{1}{2\pi I_0\left(c\right)} \exp\left[c \cos\left(x\right)\right] \; for \; - \pi < x < \pi$$

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).

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.

NextWeibull

```
virtual public double NextWeibull(double a)
```

Description

Generate a pseudorandom number from a Weibull distribution.

Method `NextWeibull` generates pseudorandom numbers from a Weibull distribution with shape parameter a. The probability density function is

$$f\left(x\right)=Ax^{A-1}e^{-x^{A}}\,for\,x\geq 0$$

`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,

$$r\left(x\right)=\frac{1}{\alpha^{2}}x\,e^{\left(-x^{2}/2\alpha^{2}\right)}\;for\,x\geq 0$$

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.

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.

Skip

```
virtual public void Skip(int n)
```

Description

Resets the seed to skip ahead in the base linear congruential generator.

This method can be used only if a linear congruential multiplier is explicitly defined by a call to Multiplier (p. 661).

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. 666) calls it twice.

Parameter

`n` – A `int` which specifies the number of random deviates to skip.

Description

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

$$x_i \equiv cx_{i-1}\text{mod}(2^{31} - 1)$$

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. 661). 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. 679) or Imsl.Stat.MersenneTwister64 (p. 683). 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. 668).

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.

```
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;
        Imsl.Stat.Random r = new Imsl.Stat.Random(123457);
        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
```

Random.BaseGenerator Interface

Summary

Base pseudorandom number.

`public interface Imsl.Stat.Random.BaseGenerator`

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()
```

NextFloat

```
abstract public float NextFloat()
```

MersenneTwister Class

Summary

A 32-bit Mersenne Twister generator.

```
public class Imsl.Stat.MersenneTwister :  Imsl.Stat.Random.BaseGenerator,
ICloneable
```

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(System.UInt32 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(System.UInt32[] 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.

Only the first 32 bits of the `double` are pseudorandom.

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.

Description

By default, the class Imsl.Stat.Random (p. 661) uses the uniform distribution generated by the base class System.Random. Alternatively, one can instantiate Imsl.Stat.MersenneTwister (p. 679) or Imsl.Stat.MersenneTwister64 (p. 683) to generate uniform psuedorandom 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` or This generator can be used to generate non-uniform distributions by creating an Imsl.Stat.Random (p. 661) 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 the following C program.

A C-program for MT19937, with initialization improved 2002/1/26. Coded by Takuji Nishimura and Makoto Matsumoto.

Any feedback is very welcome.

http://www.math.sci.hiroshima-u.ac.jp/m̃-mat/MT/emt.html

mailto: m-mat@math.sci.hiroshima-u.ac.jp

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.

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using Imsl.Stat;

public class MersenneTwisterEx1
{
   /// <summary>
   /// The main entry point for the application.
   /// </summary>
   [STAThread]
   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();
      }

      Console.WriteLine("              First Stream Output");
```

```
      Console.WriteLine(r[0]+"     "+r[1]+"     "+r[2]+"     "+r[3]);

      /* Check the cloned copy against the original */
      Imsl.Stat.Random rndm2 = new Imsl.Stat.Random(mt2);
      for (int k=0; k < nr; k++)
      {
         r[k] = rndm2.NextDouble();
      }

      Console.WriteLine("\n              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("\n              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

Summary

A 64-bit Mersenne Twister generator.

```
public class Imsl.Stat.MersenneTwister64 :  Imsl.Stat.Random.BaseGenerator,
ICloneable
```

Constructors

MersenneTwister64

`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(System.UInt64 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(System.UInt64[] 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 System.Int64 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.

Description

`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.

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 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/m̃-mat/MT/emt.html

email: m-mat@math.sci.hiroshima-u.ac.jp

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.

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using Imsl.Stat;

public class MersenneTwister64Ex1
{
   /// <summary>
   /// The main entry point for the application.
   /// </summary>
   [STAThread]
   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");
      Console.WriteLine(r[0]+"     "+r[1]+"     "+r[2]+"     "+r[3]);

      /* Check the cloned copy against the original */
      Imsl.Stat.Random rndm2 = new Imsl.Stat.Random(mt2);
```

```
      for (int k=0; k < nr; k++)
      {
         r[k] = rndm2.NextDouble();
      }

      Console.WriteLine("\n              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 = (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();
      }
      Console.WriteLine("\n              Serialized Stream Output");
      Console.WriteLine(r[0]+"     "+r[1]+"     "+r[2]+"     "+r[3]);
   }
}
```

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

Summary

Generates the low-discrepancy Faure sequence.

`public class Imsl.Stat.FaureSequence :  Imsl.Stat.IRandomSequence`

Properties

Base

`public int Base {get; }`

Description

The base.

Dimension

`Final public int Dimension {get; }`

Description

Returns the dimension of the sequence.

Skip

`public int Skip {get; }`

Description

Returns the number of points skipped at the beginning of the sequence.

Constructors

FaureSequence

`public FaureSequence(int dimension)`

Description

Creates a Faure sequence with the default base.

The base defaults to the smallest prime equal to or greater than `dimension`.

Parameter

`dimension` – An `int` which specifies the dimension of the sequence.

FaureSequence

`public FaureSequence(int dimension, int baseSequence, int nSkip)`

Description

Creates a Faure sequence.

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.

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.

Methods

ComputeParameters

`void ComputeParameters()`

Description

Compute needed parameters.

NextDouble

`public double NextDouble()`

Description

Returns the first value of the next point in the sequence.

This method is intended for use when `dimension` is 1.

Returns

A `double` array which specifies the next sequence value.

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.

If `n` is less than or equal to 2 then 2 is returned.

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`.

Description

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

$$E = [0, t_1) \times \cdots \times [0, t_d),\ 0 \leq t_j \leq 1,\ 1 \leq j \leq d,$$

λ is the Lebesque 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

$$D_n^{(d)} \leq c(d) \frac{(\log n)^d}{n}$$

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, $0 \leq a_j(n) < b$.

The j-th coordinate of x_n is

$$x_n^{(j)} = \sum_{k=0}^{\infty} \sum_{d=0}^{\infty} c_{kd}^{(j)} a_d(n) b^{-k-1},\ 1 \leq j \leq d$$

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.

Example: FaureSequence

In this example, ten points of the Faure sequence are computed. The points are in a four-dimensional cube.

```
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

```
              Faure Sequence
      0         1         2         3
0  0.201344  0.274944  0.532544  0.694144
1  0.401344  0.474944  0.732544  0.894144
2  0.601344  0.674944  0.932544  0.094144
3  0.801344  0.874944  0.132544  0.294144
4  0.841344  0.114944  0.572544  0.934144
5  0.041344  0.314944  0.772544  0.134144
6  0.241344  0.514944  0.972544  0.334144
7  0.441344  0.714944  0.172544  0.534144
8  0.641344  0.914944  0.372544  0.734144
9  0.681344  0.154944  0.612544  0.374144
```

IRandomSequence Interface

Summary

Interface implemented by generators of random or quasi-random multidimension sequences.

`public interface Imsl.Stat.IRandomSequence`

Property

Dimension

`abstract public int Dimension {get; }`

Description

Returns the dimension of the sequence.

Method

NextPoint

`abstract public double[] NextPoint()`

Description

Returns the next multidimensional point in the sequence.

Returns

A `double` array of length *dimension*.

Chapter 22: Finance

Types

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.

`Bond.Frequency`	**Meaning**
`Bond.Annual`	One payment per year (Annual payment)
`Bond.SemiAnnual`	Two payments per year (Semi-annual payment)
`Bond.Quarterly`	Four payments per year (Quarterly payment)

The `Finance.Period` field indicates when payments are due.

`Finance.Period`	**Meaning**
`Finance.At_End_of_Period`	Payments are due at the end of the period
`Finance.AT_Beginning_of_Period`	Payments are due at the beginning of the period

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.

Class Field	Day count basis
`DayCountBasis.BasisNASD`	US (NASD) 30/360
`DayCountBasis.BasisActualActual`	Actual/Actual
`DayCountBasis.BasisActual360`	Actual/360
`DayCountBasis.BasisActual365`	Actual/365
`DayCountBasis.Basis30e360`	European 30/360

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:

- *SIA Standard Securities Calculation Methods* 1993, vols. 1 and 2, Third Edition
- *Microsoft Excel 5, Worksheet Function Reference.*

Finance Class

Summary

Collection of finance functions.

`public class Imsl.Finance.Finance`

Constructor

Finance

`public Finance()`

Description

Initializes a new instance of the Imsl.Finance.Finance (p. 696) class.

Methods

Cumipmt

```
static public double Cumipmt(double rate, int nper, double pv, int
  firstPeriod, int lastPeriod, Imsl.Finance.Finance.Period period)
```

Description

Returns the cumulative interest paid between two periods.

It is computed using the following:

$$\sum_{i=firstPeriod}^{lastPeriod} interest_i$$

where $interest_i$ is computed from `Ipmt` for the i-th period.

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 caclulation. 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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)

Returns

A `double` which specifies the cumulative interest paid between the first period and the last 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.

It is computed using the following:

$$\sum_{i=firstPeriod}^{lastPeriod} principal_i$$

where $principal_i$ is computed from `Ppmt` for the i-th period.

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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)).

Returns

A `double` which specifies the cumulative principal paid between the first period and the last 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.

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,

$$(\text{cost} - \text{total depreciation from priorperiods}) \times rate$$

where

$$rate\ = 1 - \left(\frac{\text{salvage}}{\text{cost}}\right)^{\left(\frac{1}{life}\right)}$$

NOTE: *rate* is rounded to three decimal places.

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.

Ddb

```
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.

It is computed using the following:

$$[cost - salvage\,(total\ depreciation\ from\ prior\ periods)]\ \frac{factor}{life}$$

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.

Dollarde

```
static public double Dollarde(double fractionalDollar, int fraction)
```

Description

Converts a fractional price to a decimal price.

It is computed using the following:

$$idollar + (fractionalDollar - idollar) \times \frac{10^{(ifrac+1)}}{fraction}$$

where *idollar* is the integer part of *fractionalDollar*, and *ifrac* is the integer part of $\log(fraction)$.

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.

Dollarfr

```
static public double Dollarfr(double decimalDollar, int fraction)
```

Description

Converts a decimal price to a fractional price.

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 $log(fraction)$.

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.

Effect

```
static public double Effect(double nominalRate, int nper)
```

Description

Returns the effective annual interest rate.

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:

$$\left(1 + \frac{nominalRate}{nper}\right)^{nper} - 1$$

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.

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.

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 $rate \neq 0$,

$$pv(1 + rate)^{nper} + pmt\left[1 + rate\,(period)\right]\frac{(1 + rate)^{nper} - 1}{rate} + fv = 0$$

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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)).

Returns

A `double` which specifies the future value of an investment.

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.

It is computed using the following:

$$\sum_{i=1}^{count} (principal \times schedule_i)$$

where $schedule_i$ = interest rate at the i-th period.

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

---

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.

It is computed using the following:

$$\left\{ pv\,(1+rate)^{nper-1} + pmt\,(1+rate \times period)\,\frac{(1+rate)^{nper-1}}{rate} \right\} rate$$

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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)).

Returns

A `double` which specifies the interest payment for a given period for an investment.

Irr

`static public double Irr(double[] pmt)`

Description

Returns the internal rate of return for a schedule of cash flows.

It is found by solving the following:

$$0 = \sum_{i=1}^{count} \frac{value_i}{(1 + rate)^i}$$

where $value_i$ = the ith cash flow, $rate$ is the internal rate of return.

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.

Irr

`static public double Irr(double[] pmt, double guess)`

Description

Returns the internal rate of return for a schedule of cash flows.

It is found by solving the following:

$$0 = \sum_{i=1}^{count} \frac{value_i}{(1 + rate)^i}$$

where $value_i$ = the ith cash flow, $rate$ is the internal rate of return.

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.

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.

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:

$$\left\{\frac{-\left(pnpv\right)\left(1+reinvestRate\right)^{n\text{-}per}}{\left(nnpv\right)\left(1+financeRate\right)}\right\}^{\frac{1}{nper-1}}-1$$

where *pnpv* is calculated from `Npv` for positive values in values using `reinvestRate`, and where *nnpv* is calculated from `Npv` for negative values in `values` using `financeRate`.

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.

Nominal

`static public double Nominal(double effectiveRate, int nper)`

Description

Returns the nominal annual interest rate.

The nominal interest rate is the interest rate as stated on the face of a security. It is computed using the following:

$$\left[\left(1+effectiveRate\right)^{\frac{1}{nper}}-1\right]\times nper$$

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.

Nper

```
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.

It can be found by solving the following:

If $rate = 0$,

$$pv + pmt \times nper + fv = 0$$

If $rate \neq 0$,

$$pv(1 + rate)^{nper} + pmt\left[1 + rate\left(period\right)\right]\frac{(1 + rate)^{nper} - 1}{rate} + fv = 0$$

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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)).

Returns

A `int` which specifies the number of periods for an investment.

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.

It is found by solving the following:

$$\sum_{i=1}^{count} \frac{value_i}{(1 + rate)^i}$$

where $value_i$ = the ith cash flow.

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.

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.

It can be found by solving the following:

If $rate = 0$,

$$pv + pmt \times nper + fv = 0$$

If $rate \neq 0$,

$$pv(1 + rate)^{nper} + pmt\left[1 + rate\left(period\right)\right]\frac{(1 + rate)^{nper} - 1}{rate} + fv = 0$$

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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)).

Returns

A `double` which specifies the interest payment for a given period for an investment.

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.

It is computed using the following:

$$payment_i - interest_i$$

where $payment_i$ is computed from `pmt` for the i-th period, $interest_i$ is calculated from `Ipmt` for the i-th 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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)).

Returns

A `double` which specifies the payment on the principal for a given 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.

It can be found by solving the following:

If $rate = 0$,

$$pv + pmt \times nper + fv = 0$$

If $rate \neq 0$,

$$pv(1 + rate)^{nper} + pmt\left[1 + rate\,(period)\right]\frac{(1 + rate)^{nper} - 1}{rate} + fv = 0$$

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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)).

Returns

A `double` which specifies the present value of the investment.

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.

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 $rate \neq 0$,

$$pv(1 + rate)^{nper} + pmt\,[1 + rate\,(period)]\,\frac{(1 + rate)^{nper} - 1}{rate} + fv = 0$$

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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)).

Returns

A `double` which specifies the interest rate per period of an annuity.

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.

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 $rate \neq 0$,

$$pv(1 + rate)^{nper} + pmt\left[1 + rate\left(period\right)\right] \frac{(1 + rate)^{nper} - 1}{rate} + fv = 0$$

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. 728) or Imsl.Finance.Finance.Period.AtBeginning (p. 728)).

`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.

Sln

```
static public double Sln(double cost, double salvage, int life)
```

Description

Returns the depreciation of an asset using the straight line method.

It is computed using the following:

$$cost - salvage/life$$

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.

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.

It is computed using the following:

$$(cost - salvage)(per)\ \frac{(life+1)\,(life)}{2}$$

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.

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.

It is computed using the following:

If $no\_sl = 0$,

$$\sum_{i=firstPeriod+1}^{lastPeriod} ddb_i$$

If $no\_sl \neq 0$,

$$A + \sum_{i=k}^{lastPeriod} \frac{cost - A - salvage}{lastPeriod - k + 1}$$

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.

$$A = \sum_{i=firstPeriod+1}^{k-1} ddb_i$$

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 `boolean` 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.

Xirr

`static public double Xirr(double[] pmt, System.DateTime[] dates)`

Description

Returns the internal rate of return for a schedule of cash flows.

It is not necessary that the cash flows be periodic. It can be found by solving the following:

$$0 = \sum_{i=1}^{count} \frac{value_i}{(1 + rate)^{\frac{d_i - d_1}{365}}}$$

In the equation above, d_i represents the ith payment date. d_1 represents the 1st payment date. *value* represents the ith cash flow. *rate* is the internal rate of return.

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.

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.

It is not necessary that the cash flows be periodic. It can be found by solving the following:

$$0 = \sum_{i=1}^{count} \frac{value_i}{(1 + rate)^{\frac{d_i - d_1}{365}}}$$

In the equation above, d_i represents the ith payment date. d_1 represents the 1st payment date. *value* represents the ith cash flow. *rate* is the internal rate of return.

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.

`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.

Xnpv

```
static public double Xnpv(double rate, double[] cashFlow, System.DateTime[]
  dates)
```

Description

Returns the present value for a schedule of cash flows.

It is not necessary that the cash flows be periodic. It is computed using the following:

$$\sum_{i=1}^{count} \frac{value_i}{(1 + rate)^{(d_i - d_1)/365}}$$

In the equation above, d_i represents the ith payment date, d_1 represents the first payment date, and $value_i$ represents the ith cash flow.

Parameters

`rate` – A `double` which specifies the interest rate.

`cashFlow` – A `double` array containing the cash flows.

`dates` – A `DateTime` array which contains a schedule of payment dates.

Returns

A `double` which specifies the present value.

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.

```
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.

```
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).

```
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.

```
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.

```
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.

```
using System;
using Imsl.Finance;

public 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.

```
using System;
using Imsl.Finance;

public 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 - Adustable 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.

```
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%.

```
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.

```
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.

```
using System;
using Imsl.Finance;

public class mirrEx1
{
    public static void  Main(String[] args)
    {
        double[] x = new double[]{- 4500.0, - 800.0,
                                     800.0, 800.0,
                                     600.0, 600.0,
                                     800.0, 800.0,
                                     700.0, 3000.0};
        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;

public 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%, \ncompounded 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%.

```
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%.

```
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 \nof 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.

```
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.

```
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.AtBeginning);
```

```
        Console.Out.WriteLine("The computed interest rate on the loan "
                              + "is " + rate.ToString("P"));
    }
}
```

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;
        int life = 15;
        int per = 14;

        double syd = Finance.Syd(cost, salvage, life, per);
        Console.Out.WriteLine("The depreciation allowance for the 14th"
                              +" year is " + syd.ToString("C"));
    }
}
```

Output

```
The depreciation allowance for the 14th year is $333.33
```

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.

```
using System;
using Imsl.Finance;

public class vdbEx1
{
    public 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.Out.WriteLine("The depreciation allowance between the " +
                              "10th and 15th 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% made on January 3, 1997, January 3, 1999, and January 3, 2000 is computed.

```
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

Summary

Used to indicate that payment is made at the beginning or end of each period.

`public enumeration Imsl.Finance.Finance.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

Summary

Collection of bond functions.

`public class Imsl.Finance.Bond`

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.

In the equation below, A_i represents the number of days which have accrued for the ith 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 ith quasi-coupon period within the odd Frequency. NL_i is expressed in days. `Accrint` solves the following:

$$par\left(\frac{rate}{frequency}\sum_{i=1}^{NC}\frac{A_i}{NL_i}\right)$$

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.

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.

$$= par \times rate \times \frac{A}{D}$$

In the above equation, A represents the number of days starting at issue date to maturity date and D represents the annual basis.

Parameters

`issue` – Ahe `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.

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.

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.

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.

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.

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.

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.

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.

Convexity is the sensitivity of the duration of a security to changes in yield. It is computed using the following:

$$\frac{\frac{1}{(q\times frequency)^2}\left\{\sum_{t=1}^{n} t\,(t+1)\left(\frac{coupon}{frequency}\right)q^{-t}+n\,(n+1)\,q^{-n}\right\}}{\left(\sum_{t=1}^{n}\left(\frac{coupon}{frequency}\right)q^{-t}+q^{-n}\right)}$$

where n is calculated from `Coupnum`, and $q = 1 + \frac{yield}{frequency}$.

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.

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.

For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

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.

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.

For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

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.

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.

For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

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.

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.

For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

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.

Coupnum

```
static public int Coupnum(System.DateTime settlement, System.DateTime
  maturity, Imsl.Finance.Bond.Frequency frequency, Imsl.Finance.DayCountBasis
  basis)
```

Description

Returns the number of coupons payable between the settlement date and the maturity date.

For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

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.

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.

For a good discussion on day count basis, see *SIA Standard Securities Calculation Methods* 1993, vol. 1, pages 17-35.

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.

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.

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:

$$\frac{redemption - price}{price} \times \frac{B}{DSM}$$

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.

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.

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.

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:

$$\left(\frac{\frac{\frac{DSC}{E}\times 100}{\left(1+\frac{yield}{freq}\right)^{\left(N-1+\frac{DSC}{E}\right)}} + \sum_{k=1}^{N}\left(\left(\frac{100\times coupon}{freq\times\left(1+\frac{yield}{freq}\right)^{\left(k-1+\frac{DSC}{E}\right)}}\right)\times\left(k-1+\frac{DSC}{E}\right)\right)}{\frac{100}{\left(1+\frac{yield}{freq}\right)^{N-1+\frac{DSC}{E}}} + \sum_{k=1}^{N}\left(\frac{100\times coupon}{freq\times\left(1+\frac{yield}{freq}\right)^{k-1+\frac{DSC}{E}}}\right)} \right) \times \frac{1}{freq}$$

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.

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.

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.

It is computed using the following:

$$\frac{redemption - investment}{investment} \times \frac{B}{DSM}$$

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.

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.

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.

It is computed using the following:

$$\frac{duration}{1 + \frac{yield}{frequency}}$$

where *duration* is calculated from `Mduration`.

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.

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.

It is computed using the following:

$$\frac{redemption}{\left(1+\frac{yield}{frequency}\right)^{\left(N-1+\frac{DSC}{E}\right)}}+\sum_{k=1}^{N}\frac{100\times\frac{rate}{frequency}}{\left(1+\frac{yield}{frequency}\right)^{\left(k-1+\frac{DSC}{E}\right)}}-\left(100\times\frac{rate}{frequency}\times\frac{A}{E}\right)$$

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.

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.

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.

It is computed using the following:

$$redemption - rate \times redemption \times \frac{DSM}{B}$$

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.

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.

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.

It is computed using the following:

$$\frac{100 + \left(\frac{DIM}{B} \times rate \times 100\right)}{1 + \left(\frac{DSM}{B} \times yield\right)} - \frac{A}{B} \times rate \times 100$$

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.

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.

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.

It is computed using the following:

$$\frac{redemption}{1 + \left(\frac{DSM}{B}\right) yield}$$

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.

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.

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.

It is computed using the following:

$$\frac{investment}{1 - \left(rate \times \frac{DIM}{B}\right)}$$

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.

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.

Tbilleq

```
static public double Tbilleq(System.DateTime settlement, System.DateTime
  maturity, double rate)
```

Description

Returns the bond-equivalent yield of a Treasury bill.

It is computed using the following:

If $DSM <= 182$

$$\frac{365 \times rate}{360 - rate \times DSM}$$

otherwise,

$$\frac{-\frac{DSM}{365} + \sqrt{\left(\frac{DSM}{365}\right)^2 - \left(2 \times \frac{DSM}{365} - 1\right) \times \frac{rate \times DSM}{rate \times DSM - 360}}}{\frac{DSM}{365} - 0.5}$$

In the above equation, DSM represents the number of days starting at settlement date to maturity date.

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.

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.

It is computed using the following:

$$100\left(1 - \frac{rate \times DSM}{360}\right)$$

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).

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.

Tbillyield

```
static public double Tbillyield(System.DateTime settlement, System.DateTime
  maturity, double price)
```

Description

Returns the yield of a Treasury bill.

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).

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.

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.

It is computed using the following:

$$A/D$$

where A equals the number of days from `start` to `end`, D equals annual basis.

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.

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.

If there is one coupon period use the following:

$$\frac{\left(\frac{redemption}{100}+\frac{rate}{frequency}\right)-\left[\frac{price}{100}+\left(\frac{A}{E}\times\frac{rate}{frequency}\right)\right]}{\frac{price}{100}+\left(\frac{A}{E}\times\frac{rate}{frequency}\right)}\times\frac{frequency\times E}{DSR}$$

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:

$$price-\frac{redemption}{\left(\frac{1+yield}{frequency}\right)^{\frac{N-1+DSC}{E}}}-\left(\sum_{k=1}^{N}\frac{100\times\frac{rate}{frequency}}{\left(\frac{1+yield}{frequency}\right)^{\frac{k-1+DSC}{E}}}\right)+100\times\frac{rate}{frequency}\times\frac{A}{E}=0$$

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.

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.

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.

It is computed using the following:

$$\frac{redemption - price}{price} \times \frac{B}{DSM}$$

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.

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.

Yieldmat

```
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.

It is computed using the following:

$$\frac{\left[1 + \left(\frac{DIM}{B} \times rate\right)\right] - \left[\frac{price}{100} + \left(\frac{A}{B} \times rate\right)\right]}{\frac{price}{100} + \left(\frac{A}{B} \times rate\right)} \times \frac{B}{DSM}$$

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.

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.

Description

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 aquires 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 ommitted.

- price = Pricedisc(rate) (p. 739)

- price = Priceyield(yield) (p. 740)
- price = Pricemat(issue, rate, yield) (p. 739)
- rate = Disc(price) (p. 735)
- yield = Yielddisc(price) (p. 745)

A related method is Accrintm (p. 730), 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. 742)
- yield = Tbillyield(Price) (p. 743)
- yield = Tbilleq(rate) (p. 741)

Interest Paying Bonds

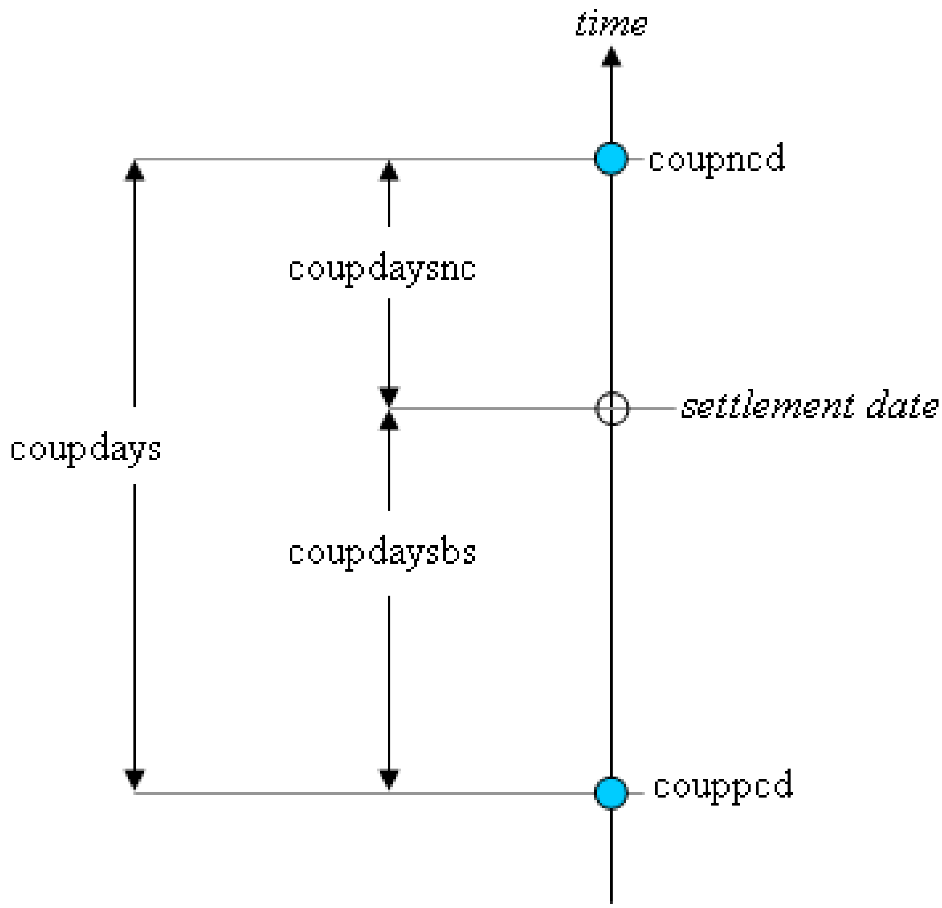
Most bonds pay interest periodically. The interest paying bond methods all have *settlement*, *maturity*, *basis* and *frequency* as arguments. Again supressing the common arguments,

- price = Price(rate, yield, redemption) (p. 738)
- yield = Yield(rate, Price, redemption) (p. 744)
- redemption = Received(Price, rate) (p. 741)

A related method is Accrint (p. 729), 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.



- Coupppcd (p. 735) is the coupon date immediately prior to the settlement date.
- Coupncd (p. 734) is the coupon date immediately after the settlement date.
- Coupdaybs (p. 732) is the number of days from the immediately prior coupon date to the settlement date.
- Coupdaysnc (p. 733) is the number of days from the settlement date to the next Coupon date.
- Coupdays (p. 733) is the number of days between these two coupon dates.

A related method is Coupnum (p. 734), which returns the number of coupons payable between settlement and maturity.

Another related method is Yearfrac (p. 743), 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. 736)
- DayCountBasis modified duration (p. 737)
- Convexity (p. 731)

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.

```
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.

```
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"
                              + "\ncoupon 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.

```
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.

```
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 \nsettlement 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 \ndate 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.

```
using System;
using Imsl.Finance;

public class durationEx1
{
    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 duration = Bond.Duration(settlement, maturity, coupon,
                                        yield, freq, dcb);
        Console.Out.WriteLine("The annual duration of the bond with" +
                              "\nsemiannual 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.

```
using System;
using Imsl.Finance;

public 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 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.

```
using System;
using Imsl.Finance;

public 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.

priceyieldEx1

```
using System;
using Imsl.Finance;

public 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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
using System;
using Imsl.Finance;

public 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.Out.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.

```
using System;
using Imsl.Finance;

public 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.Out.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

Summary

Frequency of the bond's coupon payments.

```
public enumeration Imsl.Finance.Bond.Frequency
```

Fields

```
Annual
public Imsl.Finance.Bond.Frequency Annual
```

Description

Indicates interest is paid once 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

Summary

The Day Count Basis.

```
public class Imsl.Finance.DayCountBasis
```

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. 766)

```
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.

See Also: Imsl.Finance.DayCountBasis.BasisPartActual (p. 767), Imsl.Finance.DayCountBasis.BasisPartNASD (p. 767)

`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.

See Also: Imsl.Finance.DayCountBasis.BasisPartActual (p. 767), Imsl.Finance.DayCountBasis.BasisPart365 (p. 766)

`BasisActualActual`

`public Imsl.Finance.DayCountBasis BasisActualActual`

Description

Computations are based on the actual calendar.

See Also: Imsl.Finance.DayCountBasis.BasisPartActual (p. 767)

`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. 767)

`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; }`

Description

The (days in month) portion of the Day Count Basis.

YearBasis

`public Imsl.Finance.IBasisPart YearBasis {get; }`

Description

The (days in year) portion of the Day Count Basis.

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.

Description

Rules for computing the number or 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.

IBasisPart Interface

Summary

Component of DayCountBasis.

`public interface Imsl.Finance.IBasisPart`

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 settlement,
  System.DateTime maturity)
```

Description

Returns the number of days in the year.

Parameters

`settlement` – A `DateTime` object containing the settlement date.

`maturity` – A `DateTime` object containing the maturity date.

Returns

A `int` which specifies the number of days in the year.

Description

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. 765)

Chapter 23: Chart2D

Types

AbstractChartNode Class

Summary

The base class of all of the nodes in 2D chart trees.

`public class Imsl.Chart2D.AbstractChartNode`

Fields

`AUTOSCALE_DATA`
`public int AUTOSCALE_DATA`

Description

An `int` that indicates autoscaling is to be done by scanning the data nodes.

See Also: Imsl.Chart2D.AbstractChartNode.AutoscaleInput (p. 776)

AUTOSCALE_DENSITY

`public int AUTOSCALE_DENSITY`

Description

An `int` that indicates autoscaling is to adjust the "Density" attribute.

This applies only to time axes.

See Also: Imsl.Chart2D.AbstractChartNode.AutoscaleOutput (p. 777)

AUTOSCALE_NUMBER

`public int AUTOSCALE_NUMBER`

Description

An `int` that indicates autoscaling is to adjust the "Number" attribute.

See Also: Imsl.Chart2D.AbstractChartNode.AutoscaleOutput (p. 777)

AUTOSCALE_OFF

`public int AUTOSCALE_OFF`

Description

An `int` that indicates autoscaling is turned off.

See Also: Imsl.Chart2D.AbstractChartNode.AutoscaleInput (p. 776), Imsl.Chart2D.AbstractChartNode.AutoscaleOutput (p. 777)

AUTOSCALE_WINDOW

`public int AUTOSCALE_WINDOW`

Description

An `int` that indicates autoscaling is to be done by using the "Window" attribute.

See Also: Imsl.Chart2D.AbstractChartNode.AutoscaleInput (p. 776), Imsl.Chart2D.AbstractChartNode.AutoscaleOutput (p. 777)

AXIS_X

`public int AXIS_X`

Description

An `int` that indicates the x-axis.

See Also: Type (p. 819)

AXIS_Y

`public int AXIS_Y`

Description

An `int` that indicates the y-axis.

See Also: Type (p. 819)

`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 the an element is not to be labeled.

See Also: Imsl.Chart2D.AbstractChartNode.LabelType (p. 779)

`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. 779)

`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. 779)

`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. 779)

`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. 781)

`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. 781)

`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. 781)

`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`.

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; }`

Description

Indicates what inputs are used for autoscaling.

Legal values are:

Value	Behavior
AUTOSCALE_OFF	Do not do autoscaling.
AUTOSCALE_DATA	Use the data values. This is the default.
AUTOSCALE_WINDOW	Use the "Window" attribute value.

AutoscaleMinimumTimeInterval

```
virtual public int AutoscaleMinimumTimeInterval {get; set; }
```

Description

Specifies the minimum tick mark interval for autoscaled time axes.

Legal values are:

AbstractChartNode.MILLISECONDAbstractChartNode.SECONDAbstractChartNode.MINUTEAbstractChartN

AutoscaleOutput

```
virtual public int AutoscaleOutput {get; set; }
```

Description

Specifies what attributes to change as a result of autoscaling.

Legal values are bitwise-or combinations of the following:

Value	Behavior
AUTOSCALE_OFF	Do not do autoscaling.
AUTOSCALE_WINDOW	Change the "Window" attribute value.
AUTOSCALE_NUMBER	Change the "Number" attribute value.
AUTOSCALE_DENSITY	Change the "Density" attribute value.

The default is (AUTOSCALE_NUMBER — AUTOSCALE_WINDOW — AUTOSCALE_DENSITY).

CultureInfo

```
virtual public System.Globalization.CultureInfo CultureInfo {get; set; }
```

Description

Adds support for Windows supported locales.

Default: CurrentCulture (p. ??)

CustomTransform

```
virtual public Imsl.Chart2D.Transform CustomTransform {get; set; }
```

Description

Allows for the specification of a custom transform.

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.

Default: 4

FillColor

`virtual public System.Drawing.Color FillColor {get; set; }`

Description

Specifies a color that will be used to fill an area.

Default: `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.

FontName

`virtual public string FontName {get; set; }`

Description

Specifies the font to be used by name.

Default: Sanserif

FontSize

`virtual public float FontSize {get; set; }`

Description

Specifies the font size.

Default: 8.

FontStyle

`virtual public System.Drawing.FontStyle FontStyle {get; set; }`

Description

Specifies the font style to be used.

Default: FontStyle.Regular (p. ??).

ImageAttr

`virtual public System.Drawing.Image ImageAttr {get; set; }`

Description

An image that is to rendered when this `ChartNode` is displayed.

IsVisible

`virtual public bool IsVisible {get; set; }`

Description

Specifies if this node and its children will be drawn.

If `false`, this node and its children are not drawn. Default: `true`.

LabelType

`virtual public int LabelType {get; set; }`

Description

Specifies the type of label to display.

This indicates how a data point is to be labeled. The default is to not label data points, `LABEL_TYPE_NONE`.

See Also:
Imsl.Chart2D.AbstractChartNode.LABEL$_TYPE_NONE(p.774), Imsl.Chart2D.AbstractChartNode.LABEL_TYPE_T$

LineColor

`virtual public System.Drawing.Color LineColor {get; set; }`

Description

Specifies the line color for this node.

Default: `Color.Black`

LineWidth

`virtual public double LineWidth {get; set; }`

Description

Specifies the line width for this node.

Default: 1.0

MarkerColor

`virtual public System.Drawing.Color MarkerColor {get; set; }`

Description

Specifies what color will be used when rendering marker.

Default: `Color.Black`.

MarkerSize

`virtual public double MarkerSize {get; set; }`

Description

Specifies the size of markers.

Default: 1.0.

Name

`virtual public string Name {get; set; }`

Description

Specifies the name of this node.

Number

`virtual public int Number {get; set; }`

Description

Specifies the number of tick marks along an axis.

Default: 0

SkipWeekends

`virtual public bool SkipWeekends {get; set; }`

Description

Specifies whether to skip weekends.

Default: `false`.

See Also: Imsl.Chart2D.AbstractChartNode.AutoscaleMinimumTimeInterval (p. 777)

TextColor

`virtual public System.Drawing.Color TextColor {get; set; }`

Description

Specifies the text color.

The default value is `Color.Black`.

TextFormat

`virtual public string TextFormat {get; set; }`

Description

Specifies the "TextFormat" attribute value.

The default is "0.00" that allows exactly two digits after the decimal.

TextFormatProvider

`virtual public System.IFormatProvider TextFormatProvider {get; set; }`

Description

Specifies the "TextFormatProvider" attribute value.

The default is `null`.

TickLength

`virtual public double TickLength {get; set; }`

Description

This scales the length of the tick mark lines.

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. Default: 1.0.

Transform

`virtual public int Transform {get; set; }`

Description

Specifies whether the axis is linear, logarithmic or a custom transform.

Legal values are
Imsl.Chart2D.AbstractChartNode.TRANSFORM$_{L}INEAR(p.775)(thedefault), Imsl.Chart2D.AbstractChartNode.T$

Constructor

AbstractChartNode

`public AbstractChartNode(Imsl.Chart2D.AbstractChartNode parent)`

Description

This interface contains members that will be common to chart objects in a variety of dimentions.

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.

The value of an attribute is returned if it is defined and its value is of type `bool`. Otherwise the `defaultValue` is returned.

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.

GetColorAttribute

`virtual public System.Drawing.Color GetColorAttribute(string name)`

Description

Convenience routine to get a Color-valued attribute.

The value of an attribute is returned if it is defined and its value is of type `Color`. Otherwise the `Color.Black` is returned.

Parameter

`name` – A `String` which contains the name of the attribute to be assessed.

Returns

A `Color` containing the attribute value.

GetDoubleAttribute

`virtual public double GetDoubleAttribute(string name, double defaultValue)`

Description

Convenience routine to get a Double-valued attribute.

The value of an attribute is returned if it is defined and its value is of type `double`. Otherwise the `defaultValue` is returned.

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.

GetIntegerAttribute

`virtual public int GetIntegerAttribute(string name, int defaultValue)`

Description

Convenience routine to get an Integer-valued attribute.

The value of an attribute is returned if it is defined and its value is of type `int`. Otherwise the `defaultValue` is returned.

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.

GetStringAttribute

`virtual public string GetStringAttribute(string name)`

Description

Convenience routine to get a String-valued attribute.

The attribute value is returned if it is defined and its value is of type `String`.

Parameter

`name` – A `String` which contains the name of the attribute to be assessed.

Returns

The `String` value of the attribute.

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).

The definition must have been set directly in this node, not just inherited from its parent node.

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.

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

Summary

The base class of all of the nodes in the 2D chart tree.

`public class Imsl.Chart2D.ChartNode : AbstractChartNode`

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. 794)

BAR_TYPE_VERTICAL
`public int BAR_TYPE_VERTICAL`

Description

Flag to indicate a vertical bar chart.

See Also: Imsl.Chart2D.ChartNode.BarType (p. 794)

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. 804)

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. 804)

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. 804)

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. 804)

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.

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.

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.

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".

This constant may be used with the Imsl.Chart2D.ChartNode.FillType (p. 796) property.

See Also:
Imsl.Chart2D.ChartNode.SetGradient(System.Drawing.Color,System.Drawing.Color,System.Drawing.Color,System.Dr
(p. 803), Imsl.Chart2D.ChartNode.GetGradient (p. 800)

`FILL_TYPE_NONE`
`public int FILL_TYPE_NONE`

Description

An `int` which indicates that a region is not to be drawn.

When Imsl.Chart2D.ChartNode.FillType (p. 796) and Imsl.Chart2D.ChartNode.FillOutlineType (p. 796) 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.

See Also: Imsl.Chart2D.ChartNode.SetFillPaint(System.Drawing.Brush) (p. 803), Imsl.Chart2D.ChartNode.GetFillPaint (p. 800)

`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. 796) and Imsl.Chart2D.ChartNode.FillOutlineType (p. 796).

`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.

This flag only applies to pie charts.

See Also: LabelType (p. 779)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`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. 797)

`MARKER_TYPE_PLUS`

`public int MARKER_TYPE_PLUS`

Description

An `int` that indicates a plus-shaped data marker is to be drawn.

This is the default value of Imsl.Chart2D.ChartNode.MarkerType (p. 797)

`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. 797)

`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. 797)

`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. 797)

`TEXT_X_CENTER`

`public int TEXT_X_CENTER`

Description

An `int` which indicates that text should be centered.

See Also: Alignment (p. 851)

`TEXT_X_LEFT`

`public int TEXT_X_LEFT`

Description

An `int` which indicates that text should be left justified.

See Also: Alignment (p. 851)

`TEXT_X_RIGHT`

`public int TEXT_X_RIGHT`

Description

An `int` which indicates that text should be right justified.

See Also: Alignment (p. 851)

`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. 851)

`TEXT_Y_CENTER`

`public int TEXT_Y_CENTER`

Description

An `int` which indicates that text should be vertically centered.

See Also: Alignment (p. 851)

`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. 851)

`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.

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; }`

Description

Typically provides a mapping for children from the user coordinate space to the device (screen) space.

Background

`virtual public Imsl.Chart2D.Background Background {get; }`

Description

The base graphic layer displayed behind other `ChartNode` objects in the tree.

BarGap

`virtual public double BarGap {get; set; }`

Description

Specifies the gap between bars in a group.

A gap of 1.0 means that space between bars is the same as the width of an individual bar in the group. Default: 0.0

BarType

`virtual public int BarType {get; set; }`

Description

Specifies the orientation of the `BarChart`.

Legal values are
Imsl.Chart2D.ChartNode.BAR$_{T}YPE_{V}ERTICAL(p.787)orImsl.Chart2D.ChartNode.BAR_{T}YPE_{H}ORIZONTAL($

BarWidth

`virtual public double BarWidth {get; set; }`

Description

The width of all of the groups of bars at each index.

Default: 0.5

Chart

`virtual public Imsl.Chart2D.Chart Chart {get; }`

Description

This is the root node of the chart tree.

ChartTitle

`virtual public Imsl.Chart2D.ChartTitle ChartTitle {get; set; }`

Description

Specifies a title for the chart.

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.

Default: `true`

DataType

`virtual public int DataType {get; set; }`

Description

Specifies how the data is to be rendered.

This should be some xor-ed combination of
Imsl.Chart2D.ChartNode.DATA$_TYPE_LINE(p.788), Imsl.Chart2D.ChartNode.DATA_TYPE_MARKER(p.788).Def$
$Imsl.Chart2D.ChartNode.DATA_TYPE_LINE(p.788)$

DoubleBuffering

`virtual public bool DoubleBuffering {get; set; }`

Description

Specifies whether double is active.

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.

The scale is proportional to the pie chart's radius. Default: 0.0

FillOutlineColor

`virtual public System.Drawing.Color FillOutlineColor {get; set; }`

Description

Specifies a color that will be used to outline this node.

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.

Default: Imsl.Chart2D.ChartNode.FILL$_TYPE_SOLID(p.789)$

FillType

`virtual public int FillType {get; set; }`

Description

Specifies a fill pattern type for this node.

Default: Imsl.Chart2D.ChartNode.FILL$_TYPE_SOLID(p.789)$

See Also:
Imsl.Chart2D.ChartNode.FILL$_TYPE_NONE(p.789), Imsl.Chart2D.ChartNode.FILL_TYPE_GRADIENT(p.788), In$

HREF

`virtual public string HREF {get; set; }`

Description

Used to specify an "activated" object in an image map.

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.

IsWebControl

`public bool IsWebControl {get; }`

Description

Indicates whether this is a web control.

Legend

`virtual public Imsl.Chart2D.Legend Legend {get; }`

Description

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.

If "MarkerThickness" is 2.0 then markers are drawn twice as thick as normal. Default: 1.0

MarkerType

`virtual public int MarkerType {get; set; }`

Description

Specifies the type of data marker to be drawn.

Default: Imsl.Chart2D.ChartNode.MARKER$_{T}YPE_{P}LUS(p.792)$

See Also:
Imsl.Chart2D.ChartNode.MARKER$_{T}YPE_{A}STERISK(p.789), Imsl.Chart2D.ChartNode.MARKER_{T}YPE_{X}(p.792$

Parent

`virtual public Imsl.Chart2D.ChartNode Parent {get; }`

Description

Indicates the parent of this `ChartNode`.

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.

In the case of a pie chart, this specifies the angle (in degrees) of the first slice. Default: 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.

This is set by the root `ChartNode`, so there is no `set ScreenAxis` accessor.

See Also: Imsl.Chart2D.Chart (p. 806)

ScreenSize

`virtual public System.Drawing.Size ScreenSize {get; set; }`

Description

Indicates the size of this `ChartNode`.

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.

TextAngle

`virtual public int TextAngle {get; set; }`

Description

An angle, in degrees, at which text is to be drawn.

Only multiples of 90 are allowed at this time. Default: 0

ToolTip

`virtual public string ToolTip {get; set; }`

Description

Text that can be displayed in the case where tool tips are used.

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

`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

`virtual public Imsl.Chart2D.ChartNode[] GetChildren()`

Description

Gets the list of child nodes.

If there are no children, a 0-length array is returned.

Returns

A `ChartNode[]` which contains the children of this node.

GetComponent

`virtual public System.Windows.Forms.Control GetComponent()`

Description

Gets the "Component" attribute value.

Returns

A `Control` that contains the "Component" attribute value.

GetConcatenatedViewport

`virtual public double[] GetConcatenatedViewport()`

Description

Returns the value of the "Viewport" attribute concatenated with the "Viewport" attributes set in its ancestor nodes.

Default: {`0.0, 1.0, 0.0, 1.0`}

Returns

A `double[4]` containing xmin, xmax, ymin and ymax.

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.

The array is of length four, containing {*colorLL*, *colorLR*, *colorUR*, *colorUL*}. Default: `null`

Returns

A `Color[4]` array which contains the color value of the "Gradient" attribute.

GetLineDashPattern

`virtual public double[] GetLineDashPattern()`

Description

Returns the "LineDashPattern" attribute value.

Returns `null` if the attribute has not been defined.

Returns

A `double[]` that contains the line "LineDashPattern" attribute value.

GetMarkerDashPattern

```
virtual public double[] GetMarkerDashPattern()
```

Description

Returns the "MarkerDashPattern" attribute value.

Returns `null` if the attribute has not been defined.

Returns

A `double[]` that contains the "MarkerDashPattern" attribute value.

GetScreenViewport

```
virtual public int[] GetScreenViewport()
```

Description

Returns the value of the "Viewport" attribute scaled by the screen size.

The value returned is scaled by the screen size containing the pixel coordinates for xmin, xmax, ymin and ymax.

Returns

An `int[4]` containing the "Viewport" attribute value.

GetTitle

```
virtual public Imsl.Chart2D.Text GetTitle()
```

Description

Returns the value of the "Title" attribute.

Returns

A `Text` which contains the "Title" attribute value.

GetViewport

```
virtual public double[] GetViewport()
```

Description

Returns the value of the "Viewport" attribute.

Default: {`0.0, 1.0, 0.0, 1.0`}

Returns

A `double[4]` containing xmin, xmax, ymin and ymax.

GetWebComponent

`virtual public System.Web.UI.WebControls.WebControl 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` whichis `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.

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.

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.Drawing.Brush brush)
```

Description

Sets the value of the "FillPaint" attribute.

Parameter

`brush` – A `Brush` which specifies the "FillPaint" attribute value.

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.

SetGradient

```
virtual public void SetGradient(System.Drawing.Color colorLL,
  System.Drawing.Color colorLR, System.Drawing.Color colorUR,
  System.Drawing.Color colorUL)
```

Description

Sets the value of the "Gradient" attribute.

This attribute defines a color gradient used to fill regions. Only two of the four colors given are actually used.

Parameter Values	Result
colorLL==*colorLR* and *colorUL*==*colorUR*	A vertical gradient is drawn.
colorLL==*colorUL* and *colorLR*==*colorUR*	A horizontal gradient is drawn.
colorLR== `null` and *colorUL*== `null`	A diagonal gradient is drawn.
colorLL== `null` and *colorUR*== `null`	A diagonal gradient is drawn.

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.

SetLineDashPattern

`virtual public void SetLineDashPattern(double[] lineDashPattern)`

Description

Sets the "LineDashPattern" attribute value.

Parameter

`lineDashPattern` – A `double[]` which specifies the line dash patten 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(Imsl.Chart2D.Text title)`

Description

Sets the value of the "Title" attribute.

Parameter

`title` – A `Text` which specifies the "Title" 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.

SetViewport

`virtual public void SetViewport(double xmin, double xmax, double ymin, double ymax)`

Description

Used to specify the viewport location.

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.

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.

SetViewport

`virtual public void SetViewport(double[] viewport)`

Description

Used to specify the viewport location.

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. The elements of *viewport* corrispond to xmin, xmax, ymin and ymax.

Parameter

`viewport` – A `double[4]` which specifies the "Viewport" attribute value.

Chart Class

Summary

The root node of the chart tree.

`public class Imsl.Chart2D.Chart : ChartNode : ICloneable`

Constructors

Chart

`public Chart()`

Description

This is the root of our tree, it has no parent.

This creates the `Chart` with a `null` component.

Chart

`public Chart(System.Windows.Forms.Control component)`

Description

This is the root of our tree, it has no parent.

This creates the `Chart` with the named `Component`.

Parameter

`component` – A `Component` that contains the chart.

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.

This creates the `Chart` drawn into the `Image`.

Parameter

`image` – An `Image` into which the `Chart` is to be drawn.

Methods

AddLegendItem

`virtual public void AddLegendItem(int type, Imsl.Chart2D.ChartNode node)`

Description

Adds a `Legend` to a `ChartNode`.

The possible legend types are:

DATA_TYPE_NONEDATA_TYPE_LINEDATA_TYPE_MARKERDATA_TYPE_FILL

Parameters

`type` – An `int` which specifies the `LegendItem` type.

`node` – A `ChartNode` to which a `Legend` is to be added.

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

`virtual public void Paint(System.Drawing.Graphics g)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`g` – A `Graphics` which is to be painted.

Paint

```
override public void Paint(Imsl.Chart2D.Draw draw)
```

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

PaintChart

```
virtual public void PaintChart(System.Drawing.Graphics graphics)
```

Description

Draw the chart using the given `Graphics` object.

Parameter

`graphics` – The `Graphics` object.

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

```
public void PrintGraphics(Object sender,
  System.Drawing.Printing.PrintPageEventArgs e)
```

Description

This method prints the chart on a single page.

The output is scaled to fill the page as much as possible while preserving the aspect ratio.

Parameters

`sender` – A `Object` that specifies the sender of an event.

`e` – A `PrintPageEventArgs` containing data for the PrintPage (p. **??**) event.

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

Parameter

–

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.

`height` – An `int` which specifies the height of the output image.

Description

This chart node creates the following child nodes: Imsl.Chart2D.Background (p. 810), Imsl.Chart2D.ChartTitle (p. 811) and Imsl.Chart2D.Legend (p. 812).

Background Class

Summary

The background of a chart.

`public class Imsl.Chart2D.Background : AxisXY`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

Grid is created by Imsl.Chart2D.Chart (p. 806) as its child. It can be retrieved using the method Imsl.Chart2D.ChartNode.Background (p. 794).

Fill attributes (specified with FillType (p. 796)) in this node control the drawing of the background.

ChartTitle Class

Summary

The main title of a chart.

`public class Imsl.Chart2D.ChartTitle : AxisXY`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`ChartTitle` is created by Chart (p. 806) as its child. It can be retrieved using the method ChartTitle (p. 811).

The axis title is the "Title" attribute value at this node. Text attributes (specified with CultureInfo, NumberFormatInfo.CurrentInfo (p. **??**) and DateTimeFormatInfo.CurrentInfo (p. **??**) members) in this node control the drawing of the title.

Grid Class

Summary

Draws the grid lines perpendicular to an axis.

`public class Imsl.Chart2D.Grid : ChartNode`

Property

Type

`virtual public int Type {get; }`

Description

Specifies the type of `Axis1D`.

The Axis types are:
Imsl.Chart2D.AbstractChartNode.AXIS$_X$(p.773)$Imsl.Chart2D.AbstractChartNode.AXIS_Y$(p.773)$Imsl.Chart2D.C$

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`Grid` is created by Imsl.Chart2D.Axis1D (p. 818) as its child. It can be retrieved using the Imsl.Chart2D.Axis1D.Grid (p. 819) property.

Line attributes (specified with LineColor (p. 779), LineWidth (p. 779) and SetMarkerDashPattern (p. 804)) in this node control the drawing of the grid lines.

Legend Class

Summary

A Imsl.Chart2D.Chart (p. 806) legend.

`public class Imsl.Chart2D.Legend :  AxisXY`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`Legend` is created by `Chart` as its child. It can be retrieved using the Imsl.Chart2D.ChartNode.Legend (p. 797) property.

By default the legend is not drawn. To have it drawn, set `chart.IsVisisble = true;`

Imsl.Chart2D.Data (p. 836) objects that have their "Title" attribute defined are automatically entered into the legend.

The drawing of the background of the legend box is controlled by the Fill attributes (specified with FillType (p. 796)) in this node. Text attributes (specified with CultureInfo, NumberFormatInfo.CurrentInfo (p. **??**) and DateTimeFormatInfo.CurrentInfo (p. **??**) members) in this node control the drawing of the text strings in the box.

Axis Class

Summary

The `Axis` node provides the mapping for all of its children from the user coordinate space to the device (screen) space.

`public class Imsl.Chart2D.Axis : ChartNode`

Constructor

Axis

`Axis(Imsl.Chart2D.Chart chart)`

Description

Contructs an `Axis` node.

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.

Parameter

`chart` – A `Chart` object which is the parent of this 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.

MapUserToDevice

```
abstract 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.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` object which specifies the chart tree to be rendered on the screen.

SetUpMapping

```
abstract public void SetUpMapping()
```

Description

Initializes the mappings between user and coordinate space.

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

Summary

The axes for an x-y chart.

`public class Imsl.Chart2D.AxisXY : Axis`

Properties

AxisX

`virtual public Imsl.Chart2D.Axis1D AxisX {get; }`

Description

The X axis associated with this node.

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.

The Y axis is a child of this node.

Constructor

AxisXY

`public AxisXY(Imsl.Chart2D.Chart chart)`

Description

Creates an `AxisXY`.

This also creates two `Axis1D` nodes as children of this node. They hold the decomposed mapping. The "Viewport" attributute for this node is set to [0.2,0.8] by [0.2,0.8].

Parameter

`chart` – A `Chart` which is the parent of this node.

Methods

GetCross

```
virtual public double[] GetCross()
```

Description

Returns the "Cross" attribute value.

The value is the point where the X and Y axes intersect, (*xcross*,*ycross*). If "Cross" is not defined then `null` is returned.

Returns

A `double[2]` containing the "Cross" attribute value.

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.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetCross

```
virtual public void SetCross(double[] cross)
```

Description

Sets the "Cross" attribute value.

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.

Parameter

`cross` – a `double[2]` containing the x and y-coordinate where the axes cross.

SetCross

```
virtual public void SetCross(double xcross, double ycross)
```

Description

Sets the "Cross" attribute value.

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.

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.

SetUpMapping

```
override public void SetUpMapping()
```

Description

Initializes the mappings between user and coordinate space.

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

`window` – A `double[2]` containing the "Window" attribute value.

Description

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).

Axis1D Class

Summary

An x-axis or a y-axis.

```
public class Imsl.Chart2D.Axis1D : ChartNode
```

Properties

AxisLabel

```
virtual public Imsl.Chart2D.AxisLabel AxisLabel {get; }
```

Description

The label node of this `Axis1D`.

This is a child of the axis node.

AxisLine

```
virtual public Imsl.Chart2D.AxisLine AxisLine {get; }
```

Description

The line node of this `Axis1D`.

This is a child of the axis node.

AxisTitle

```
virtual public Imsl.Chart2D.AxisTitle AxisTitle {get; }
```

Description

The title node of this `Axis1D`.

This is a child of the axis node.

AxisUnit

```
virtual public Imsl.Chart2D.AxisUnit AxisUnit {get; }
```

Description

The unit node of this `Axis1D`.

This is a child of the axis node.

FirstTick

```
virtual public double FirstTick {get; set; }
```

Description

This indicates the location of the first tick.

Default: GetWindow()[0].

Grid

`virtual public Imsl.Chart2D.Grid Grid {get; }`

Description

The grid node of this `Axis1D`.

This is a child of the axis node.

MajorTick

`virtual public Imsl.Chart2D.MajorTick MajorTick {get; }`

Description

The major tick node of this `Axis1D`.

This is a child of the axis node.

MinorTick

`virtual public Imsl.Chart2D.MinorTick MinorTick {get; }`

Description

The minor tick node of this `Axis1D`.

This is a child of the axis node.

TickInterval

`virtual public double TickInterval {get; set; }`

Description

The tick interval node of this `Axis1D`.

Type

`virtual public int Type {get; set; }`

Description

Specifies the type of this `Axis1D`.

The node types are:

Imsl.Chart2D.AbstractChartNode.AXIS$_X$(p.773)$Imsl.Chart2D.AbstractChartNode.AXIS_Y$(p.773)$Imsl.Chart2D.C$

Methods

GetTicks

```
virtual public double[] GetTicks()
```

Description

Returns the ”Ticks” attribute value.

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”.

Returns

A `double[]` containing the ”Ticks” attribute value.

GetWindow

```
virtual public double[] GetWindow()
```

Description

Returns the window for an `AxisiD`.

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.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetTicks

```
virtual public void SetTicks(double[] ticks)
```

Description

Sets the ”Ticks” attribute value.

Parameter

`ticks` – A `double[]` which contains the location, in user coordinates, of the major tick marks.

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.

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.

Description

`Axis1D` is created by Imsl.Chart2D.AxisXY (p. 815) as its child. It can be retrieved using the method Imsl.Chart2D.AxisXY.AxisX (p. 815) or Imsl.Chart2D.AxisXY.AxisY (p. 815).

It in turn creates the following child nodes: Imsl.Chart2D.Axis1D.AxisLine (p. 818), Imsl.Chart2D.Axis1D.AxisLabel (p. 818), Imsl.Chart2D.Axis1D.AxisTitle (p. 818), Imsl.Chart2D.Axis1D.AxisUnit (p. 818), Imsl.Chart2D.Axis1D.MajorTick (p. 819), Imsl.Chart2D.Axis1D.MinorTick (p. 819) and Imsl.Chart2D.Axis1D.Grid (p. 819).

The number of tick marks (”Number” attribute) is set to 5, but autoscaling can change this value.

AxisLabel Class

Summary

The labels on an axis.

`public class Imsl.Chart2D.AxisLabel :  ChartNode`

Methods

GetLabels

`virtual public Imsl.Chart2D.Text[] GetLabels()`

Description

Returns the "Labels" attribute.

Default: `null`

Returns

A `Text[]` containing the axis labels.

Paint

```
override public void Paint(Imsl.Chart2D.Draw draw)
```

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` containing the object to be painted.

SetLabels

```
virtual public void SetLabels(string[] value)
```

Description

Sets the axis label values for this node to be used instead of the default numbers.

The attribute "Number" is also set to `value.Length`.

Parameter

`value` – A `String[]` specifying the labels for the major tick marks.

Description

`AxisLabel` is created by Imsl.Chart2D.Axis1D (p. 818) as its child. It can be retrieved using the method Imsl.Chart2D.Axis1D.AxisLabel (p. 818).

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 (p. **??**) and DateTimeFormatInfo.CurrentInfo (p. **??**) members) in this node control the drawing of the axis labels.

AxisLine Class

Summary

The axis line.

`public class Imsl.Chart2D.AxisLine : ChartNode`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` with the object to be painted.

Description

`AxisLine` is created by Imsl.Chart2D.Axis1D (p. 818) as its child. It can be retrieved using the method Imsl.Chart2D.Axis1D.AxisLine (p. 818).

Line attributes (specified with LineColor (p. 779), LineWidth (p. 779) and SetMarkerDashPattern (p. 804)) in this node control the drawing of the axis line.

AxisTitle Class

Summary

The title on an axis.

`public class Imsl.Chart2D.AxisTitle : ChartNode`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`AxisTitle` is created by Imsl.Chart2D.Axis1D (p. 818) as its child. It can be retrieved using the method Imsl.Chart2D.Axis1D.AxisTitle (p. 818).

The axis title is the "Title" attribute value at this node. Text attributes (specified with CultureInfo, NumberFormatInfo.CurrentInfo (p. **??**) and DateTimeFormatInfo.CurrentInfo (p. **??**) members) in this node control the drawing of the axis title.

AxisUnit Class

Summary

The unit on an axis.

`public class Imsl.Chart2D.AxisUnit : ChartNode`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`AxisUnit` is created by Imsl.Chart2D.Axis1D (p. 818) as its child. It can be retrieved using the method Imsl.Chart2D.Axis1D.AxisUnit (p. 818).

The axis title is the "Title" attribute value at this node. Text attributes (specified with CultureInfo, NumberFormatInfo.CurrentInfo (p. **??**) and DateTimeFormatInfo.CurrentInfo (p. **??**) members) in this node control the drawing of the unit title.

MajorTick Class

Summary

The major tick marks.

`public class Imsl.Chart2D.MajorTick : ChartNode`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`MajorTick` is created by Imsl.Chart2D.Axis1D (p. 818) as its child. It can be retrieved using the Imsl.Chart2D.Axis1D.MajorTick (p. 819) property.

Line attributes (specified with LineColor (p. 779), LineWidth (p. 779) and SetMarkerDashPattern (p. 804)) in this node control the drawing of the major tick marks.

MinorTick Class

Summary

The minor tick marks.

`public class Imsl.Chart2D.MinorTick : ChartNode`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`MinorTick` is created by Imsl.Chart2D.Axis1D (p. 818) as its child. It can be retrieved using the Imsl.Chart2D.Axis1D.MinorTick (p. 819) property.

Line attributes (specified with LineColor (p. 779), LineWidth (p. 779) and SetMarkerDashPattern (p. 804)) in this node control the drawing of the minor tick marks.

Transform Interface

Summary

Defines a custom transformation along an axis.

`public interface Imsl.Chart2D.Transform`

Methods

MapUnitToUser

`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

`abstract public double MapUserToUnit(double user)`

Description

Maps user coordinates to the interval [0,1].

The user coordinate interval is specified by the "Window" attribute for the axis with which the transform is associated.

Parameter

`user` – A `double` which contains a location in user coordinates to be converted to unit coordinates.

SetupMapping

`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.

Description

`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.

TransformDate Class

Summary

Defines a transformation along an axis that skips weekend dates.

`public class Imsl.Chart2D.TransformDate : Imsl.Chart2D.Transform`

Constructor

TransformDate

`public TransformDate()`

Description

Initializes a new instance of the Imsl.Chart2D.TransformDate (p. 827) class.

Methods

IsWeekday

`virtual public bool IsWeekday(System.DateTime dateTime)`

Description

Indicates whether the specified date is a weekday.

Returns `false` if the specified day is a Saturday or Sunday.

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.

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].

The user coordinate interval is specified by the "Window" attribute for the axis with which the transform is associated.

Parameter

`user` – A `double` which contains a location in user coordinates to be converted to unit coordinates.

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

Summary

The R-axis in a polar plot.

`public class Imsl.Chart2D.AxisR : ChartNode`

Properties

AxisRLabel

`virtual public Imsl.Chart2D.AxisRLabel AxisRLabel {get; }`

Description

A `AxisRLabel` which specifies the label node associated with this axis.

AxisRLine

`virtual public Imsl.Chart2D.AxisRLine AxisRLine {get; }`

Description

Specifies the line node associated with this axis.

AxisRMajorTick

`virtual public Imsl.Chart2D.AxisRMajorTick AxisRMajorTick {get; }`

Description

Specifies the major tick associated with this axis.

This is a child of the axis node.

TickInterval

`virtual public double TickInterval {get; set; }`

Description

The tick interval node of this `AxisR`.

Window

`virtual public double Window {get; set; }`

Description

The radius at which `AxisTheta` is drawn.

The window has a maximum value of R. The R-axis always starts at 0. Default: 1.0

Methods

GetTicks

```
virtual public double[] GetTicks()
```

Description

Returns the "Ticks" attribute value.

If not set, then computed tick values are returned based on the type of axis (linear, log or custom), the attributes "Number" and "TickInterval".

Returns

A `double[]` containing the "Ticks" 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.

Description

`AxisR` is created by Imsl.Chart2D.Polar (p. 958) as its child. It can be retrieved using the Imsl.Chart2D.Polar.AxisR (p. 958).

It in turn creates the following child nodes: Imsl.Chart2D.AxisR.AxisRLine (p. 829), Imsl.Chart2D.AxisR.AxisRLabel (p. 829) and Imsl.Chart2D.AxisR.AxisRMajorTick (p. 829).

The number of tick marks ("Number" attribute) is set to 4, but autoscaling can change this value.

See Also

Imsl.Chart2D.Polar (p. 958)

AxisRLabel Class

Summary

The labels on an axis.

```
public class Imsl.Chart2D.AxisRLabel :  ChartNode
```

Methods

GetLabels

`virtual public Imsl.Chart2D.Text[] GetLabels()`

Description

Returns the "Labels" attribute.

Default: `null`

Returns

A `Text[]` containing the axis labels.

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetLabels

`virtual public void SetLabels(string[] value)`

Description

Sets the axis label values for this node to be used instead of the default numbers.

The attribute "Number" is also set to `value.Length`.

Parameter

`value` – A `String[]` specifying the labels for the major tick marks.

Description

`AxisRLabel` is created by Imsl.Chart2D.AxisR (p. 829) as its child. It can be retrieved using the method Imsl.Chart2D.AxisR.AxisRLabel (p. 829).

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 (p. ??) and DateTimeFormatInfo.CurrentInfo (p. ??) members) in this node control the drawing of the axis labels.

See Also

Imsl.Chart2D.Polar (p. 958), Imsl.Chart2D.AxisR (p. 829)

AxisRLine Class

Summary

The radius axis line in a polar plot.

`public class Imsl.Chart2D.AxisRLine : ChartNode`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`AxisRLine` is created by Imsl.Chart2D.AxisR (p. 829) as its child. It can be retrieved using the method Imsl.Chart2D.AxisR.AxisRLine (p. 829).

Line attributes (specified with LineColor (p. 779), LineWidth (p. 779) and SetMarkerDashPattern (p. 804)) in this node control the drawing of the axis line.

See Also

Imsl.Chart2D.Polar (p. 958), Imsl.Chart2D.AxisR (p. 829)

AxisRMajorTick Class

Summary

The major tick marks for the radius axis in a polar plot.

`public class Imsl.Chart2D.AxisRMajorTick : ChartNode`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`AxisRMajorTick` is created by Imsl.Chart2D.AxisR (p. 829) as its child. It can be retrieved using the method Imsl.Chart2D.AxisR.AxisRMajorTick (p. 829).

Line attributes (specified with LineColor (p. 779), LineWidth (p. 779) and SetMarkerDashPattern (p. 804)) in this node control the drawing of the major tick marks.

See Also

Imsl.Chart2D.Polar (p. 958), Imsl.Chart2D.AxisR (p. 829)

AxisTheta Class

Summary

The angular axis in a polar plot.

`public class Imsl.Chart2D.AxisTheta : ChartNode`

Methods

GetTicks

`virtual public double[] GetTicks()`

Description

Returns the ”Ticks” attribute value.

These are the positions at which the angles are labeled. The ticks are in radians, not degrees.

Returns

A `double[]` containing the ”Ticks” attribute value.

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.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetWindow

`virtual public void SetWindow(double[] window)`

Description

Sets the window for an `AxisTheta`.

The default ”Window” is [0,2pi].

Parameter

`window` – A `double` array of length two containing the angular range.

SetWindow

`virtual public void SetWindow(double min, double max)`

Description

Sets the window for an `AxisTheta`.

The default ”Window” is [0,2pi].

Parameters

`min` – A `double` which specifies the initial angular value, in radians value.

`max` – A `double` which specifies the final angular value, in radians.

Description

`AxisTheta` is created by Imsl.Chart2D.Polar (p. 958) as its child. It can be retrieved using the method Imsl.Chart2D.Polar.AxisTheta (p. 958).

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. 958)

GridPolar Class

Summary

Draws the grid lines for a polar plot.

`public class Imsl.Chart2D.GridPolar : ChartNode`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`GridPolar` is created by Imsl.Chart2D.Polar (p. 958) as its child. It can be retrieved using the Imsl.Chart2D.Polar.GridPolar (p. 958) property.

Line attributes (specified with LineColor (p. 779), LineWidth (p. 779) and SetMarkerDashPattern (p. 804)) in this node control the drawing of the grid lines.

Data Class

Summary

A data node in the chart tree.

`public class Imsl.Chart2D.Data : ChartNode`

Constructors

Data

`public Data(Imsl.Chart2D.ChartNode parent)`

Description

Creates a data node.

Parameter

`parent` – A `ChartNode` which specifies the parent of this data node.

Data

`public Data(Imsl.Chart2D.ChartNode parent, double[] y)`

Description

Creates a `Data` node with y values.

The x values are set to the double array containing {`0,1,...,y.Length-1`}.

Parameters

`parent` – A `ChartNode` which specifies the parent of this data node.

`y` – A `double` array containing the dependant values for this node.

Data

```
public Data(Imsl.Chart2D.ChartNode parent, Imsl.Chart2D.ChartFunction cf,
  double a, double b)
```

Description

Creates a `Data` node with y values.

The x values are set to the double array containing {`0,1,...,y.Length-1`}.

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.

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.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetDataRange

```
virtual public void SetDataRange(double[] range)
```

Description

Update the data range.

The entries in *range* are updated to reflect the extent of the data in this node. *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.

Parameter

`range` – A `double[4]` which contains the updated range, {xmin,xmax,ymin, ymax}.

Description

Drawing of a `Data` node is determined by the DataType (p. 795) property. Multiple bits can be set in "DataType".

If the DATA_TYPE_LINE (p. 788) bit is set, the line attributes are active.

If the DATA_TYPE_MARKER (p. 788) bit is set, the marker attributes are active.

If the DATA_TYPE_FILL (p. 787)} bit is set, the fill attributes are active.

If LabelType (p. 779) 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 (p. ??) and DateTimeFormatInfo.CurrentInfo (p. ??) members) in this node control the drawing of the title.

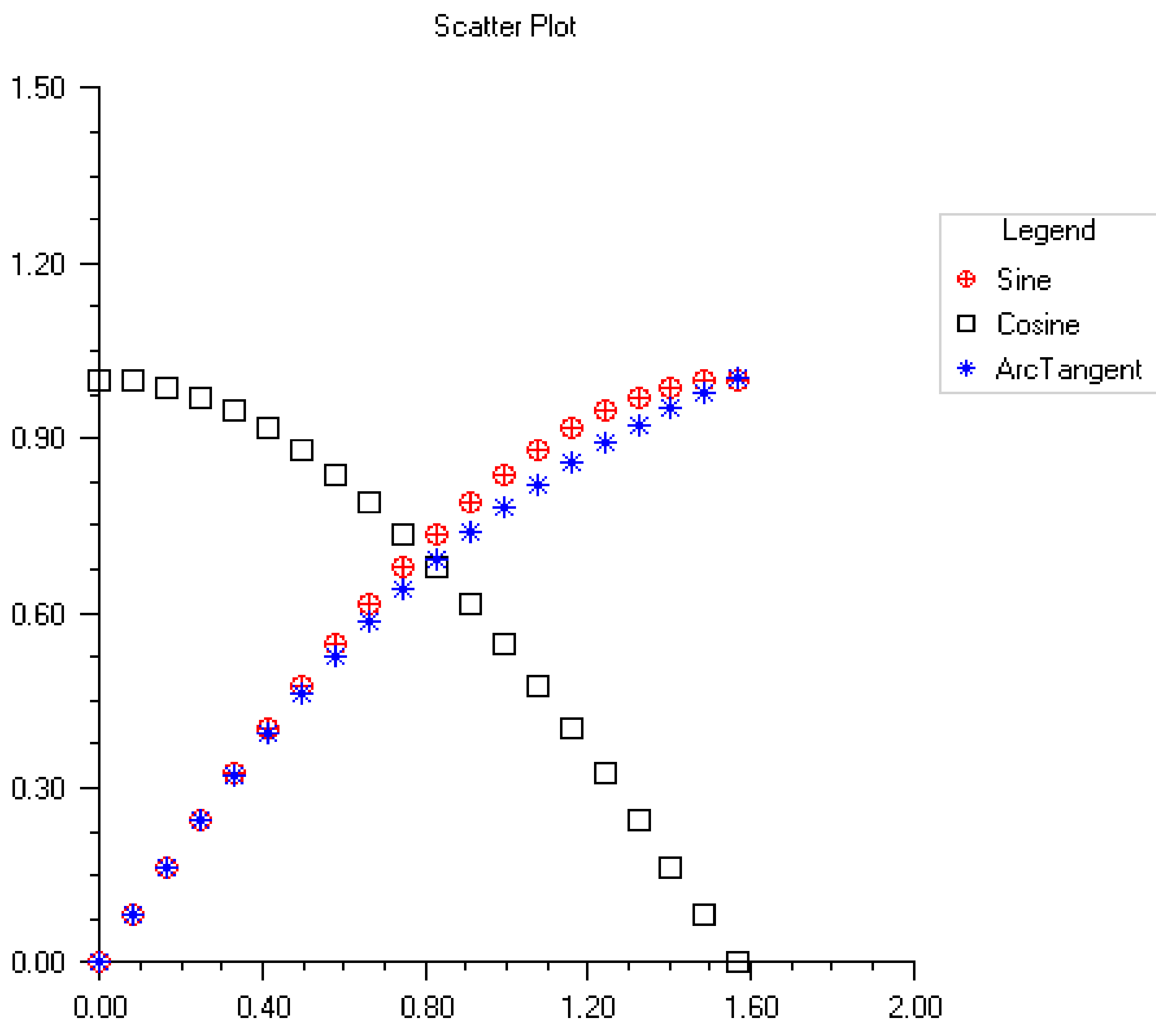
Example: Scatter Chart

A scatter plot is constructed in this example. Three data sets are used and a legend is added to the chart. This class can be used either as an applet or as an application.

```
using Imsl.Chart2D;
using Imsl.Stat;
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
        d1.MarkerColor = System.Drawing.Color.Red;
        d2.MarkerColor = System.Drawing.Color.Black;
        d3.MarkerColor = System.Drawing.Color.Blue;

        //  Set Data Labels
        d1.SetTitle("Sine");
        d2.SetTitle("Cosine");
        d3.SetTitle("ArcTangent");

        //  Add a Legend
        Legend legend = chart.Legend;
        legend.SetTitle(new Text("Legend"));
        chart.AddLegendItem(2, chart);
        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


Scatter Plot
1.50
1.20
0.90
0.60
0.30
0.00
0.00
0.40
0.80
1.20
1.60
2.00
Legend
Sine
Cosine
ArcTangent

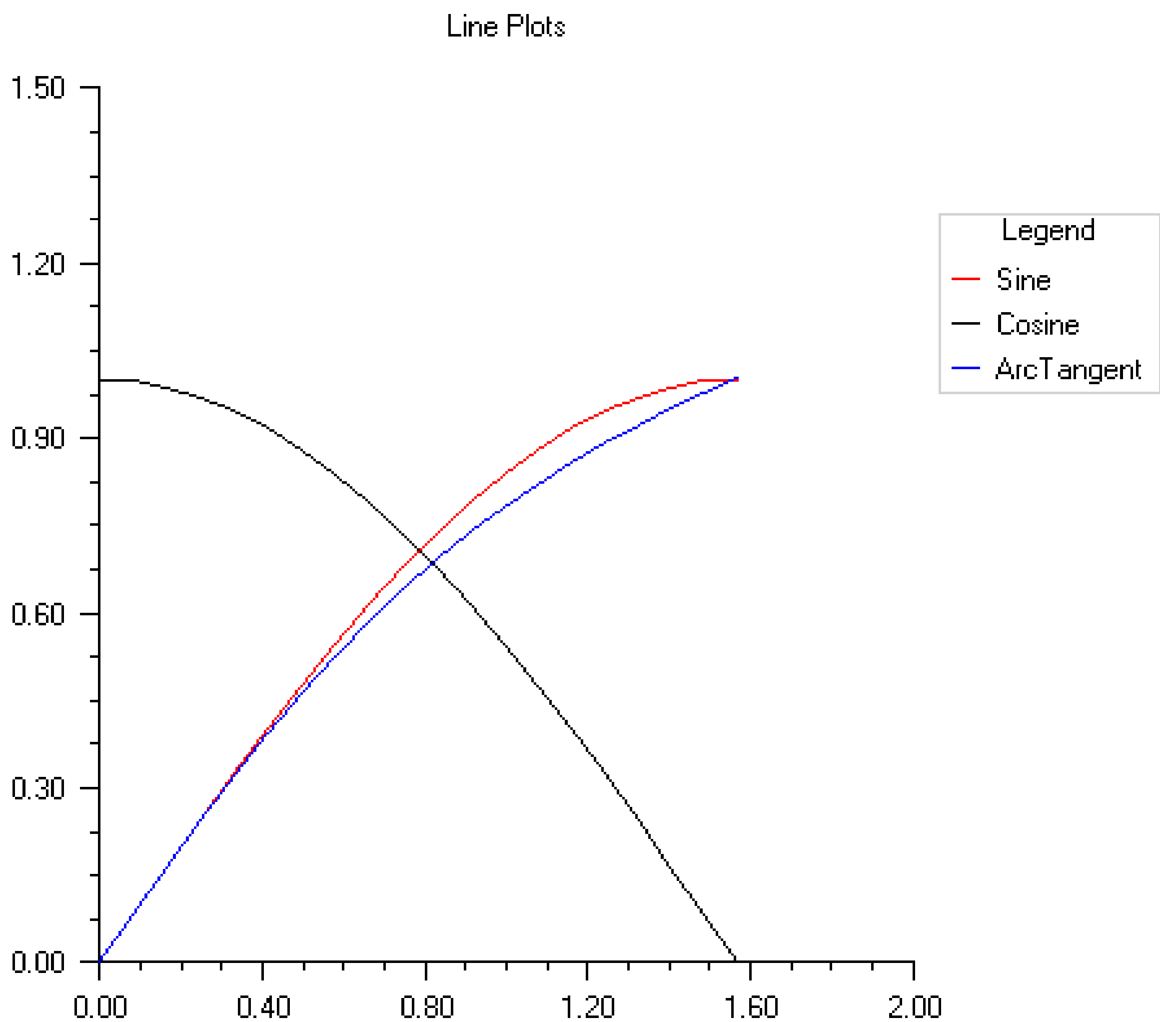
Example: Line Chart

A simple line chart is constructed in this example. Three data sets are used and a legend is added to the chart. This class can be used either as an applet or as an application.

```
using Imsl.Chart2D;
using Imsl.Stat;
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
        d1.LineColor = System.Drawing.Color.Red;
        d2.LineColor = System.Drawing.Color.Black;
        d3.LineColor = System.Drawing.Color.Blue;

        //  Set Data Labels
        d1.SetTitle("Sine");
        d2.SetTitle("Cosine");
        d3.SetTitle("ArcTangent");

        //  Add a Legend
        Legend legend = chart.Legend;
        legend.SetTitle(new Text("Legend"));
        chart.AddLegendItem(1, chart);
```

```
        legend.IsVisible = true;

        //  Set the Chart Title
        chart.ChartTitle.SetTitle("Line Plots");

    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new LineEx1());
    }
}
```

Output



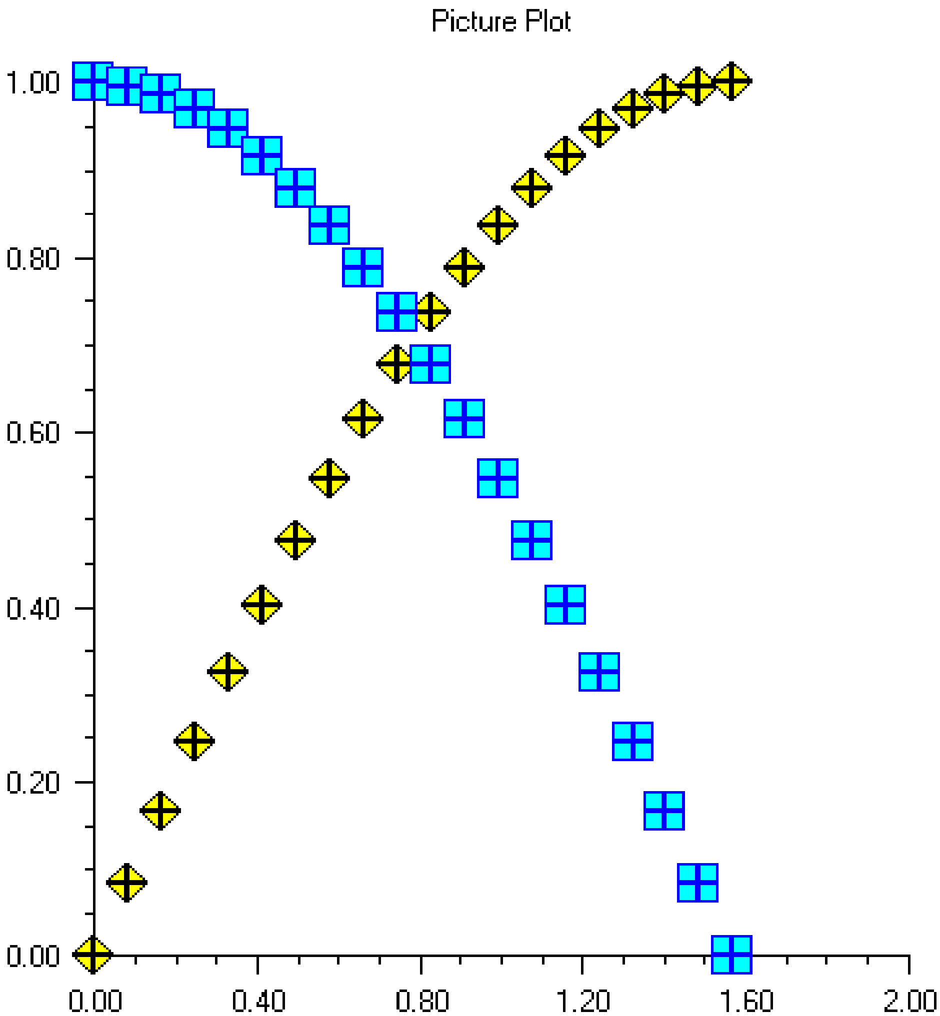
Line Plots
1.50
1.20
0.90
0.60
0.30
0.00
0.00
0.40
0.80
1.20
1.60
2.00
Legend
Sine
Cosine
ArcTangent

Example: Picture Chart

A picture plot is constructed in this example. This class can be used either as an applet or as an application.

```
using Imsl.Chart2D;
using Imsl.Stat;
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());
    }
}
```

Output



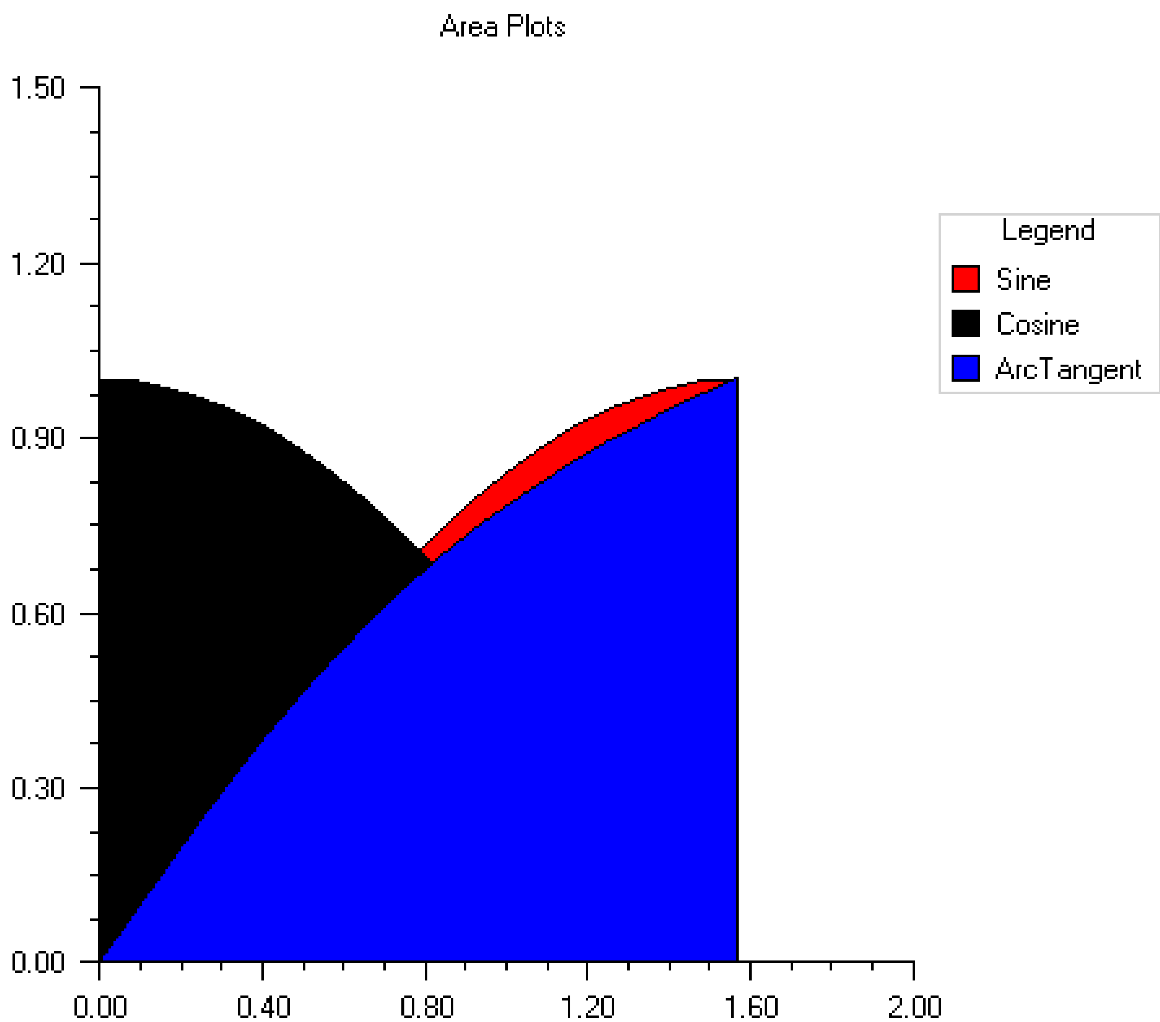
Example: Area Chart

An area chart is constructed in this example. Three data sets are used and a legend is added to the chart. This class can be used either as an applet or as an application.

```
using Imsl.Chart2D;
using Imsl.Stat;
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
        d1.LineColor = System.Drawing.Color.Red;
        d2.LineColor = System.Drawing.Color.Black;
        d3.LineColor = System.Drawing.Color.Blue;

        //  Set Fill Colors
        d1.FillColor = System.Drawing.Color.Red;
        d2.FillColor = System.Drawing.Color.Black;
        d3.FillColor = System.Drawing.Color.Blue;

        //  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());
    }
}
```

Output


Area Plots
1.50
1.20
0.90
0.60
0.30
0.00
0.00
0.40
0.80
1.20
1.60
2.00
Legend
Sine
Cosine
ArcTangent

ChartFunction Interface

Summary

An interface that allows a function to be plotted.

`public interface Imsl.Chart2D.ChartFunction`

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.

See Also

Imsl.Chart2D.Data (p. 836)

ChartSpline Class

Summary

Wraps a spline into a `ChartFunction` to be plotted.

`public class Imsl.Chart2D.ChartSpline :  Imsl.Chart2D.ChartFunction`

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`.

If zero, the function value is plotted.

If one, the first derivative is plotted, etc.

Parameters

`spline` – A `Spline` which is to have its derivative plotted.

`ideriv` – An `int` that specifies what derivative is to be plotted.

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

Summary

The value of the attribute "Title".

```
public class Imsl.Chart2D.Text
```

Properties

Alignment

`virtual public int Alignment {get; set; }`

Description

The alignment for this `Text` object.

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. 793)TEXT_X_CENTER (p. 793) TEXT_X_RIGHT (p. 793)
TEXT_Y_BOTTOM (p. 793) TEXT_Y_CENTER (p. 793) TEXT_Y_TOP (p. 793)

DefaultAlignment

`virtual public int DefaultAlignment {set; }`

Description

The default alignment for this `Text` object.

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. 793)TEXT_X_CENTER (p. 793) TEXT_X_RIGHT (p. 793)
TEXT_Y_BOTTOM (p. 793) TEXT_Y_CENTER (p. 793) TEXT_Y_TOP (p. 793)

DefaultOffset

`virtual public double DefaultOffset {set; }`

Description

The default value of the offset.

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.

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.

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.

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. 793)TEXT_X_CENTER (p. 793) TEXT_X_RIGHT (p. 793) TEXT_Y_BOTTOM (p. 793) TEXT_Y_CENTER (p. 793) TEXT_Y_TOP (p. 793)

Parameters

`text` – The `String` that is to be converted to a `Text` object.

`alignment` – An `int` which specifies the alignment.

Text

```
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 (p. ??) or DateTimeFormat (p. ??).

`obj` – A `IFormattable` that is to be converted into a `Text` object.

Description

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.

ToolTip Class

Summary

A tool tip for a chart element.

`public class Imsl.Chart2D.ToolTip :  ChartNode`

Constructor

ToolTip

`public ToolTip(Imsl.Chart2D.ChartNode parent)`

Description

Creates a `ToolTip` node that enables tool tips on charts.

Do not use the root `ChartNode` for this argument, because it will normally select only the `Background` node.

Parameter

`parent` – The `ChartNode` parent of this node.

Methods

MouseMoved

```
virtual public void MouseMoved(Object sender,
  System.Windows.Forms.MouseEventArgs e)
```

Description

The `MouseMoved` 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

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

This class requires that the chart's component be a subclass of ComponentModel (p. **??**). 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.

FillPaint Class

Summary

A collection of methods to create `Brush` objects for fill areas.

`public class Imsl.Chart2D.FillPaint`

Methods

Checkerboard

```
static public System.Drawing.Brush Checkerboard(int n, System.Drawing.Color
  colorA, System.Drawing.Color colorB)
```

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

```
static public System.Drawing.Brush Crosshatch(int n, int p,
  System.Drawing.Color colorBackground, System.Drawing.Color colorLine)
```

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,
  System.Runtime.Serialization.StreamingContext context, Object instance)
```

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,
  System.Runtime.Serialization.StreamingContext context, Object instance)
```

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

```
static public System.Drawing.Brush Diagonal(int n, System.Drawing.Color
  colorA, 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

```
static public System.Drawing.Brush Diamond(int n, int p,
  System.Drawing.Color colorBackground, System.Drawing.Color colorLine)
```

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

```
static public System.Drawing.Brush DiamondHatch(int n, int p,
  System.Drawing.Color colorBackground, System.Drawing.Color colorLine)
```

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

```
static public System.Drawing.Brush Dot(int n, int r, System.Drawing.Color
  colorBackground, System.Drawing.Color colorCircle)
```

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

```
static public System.Drawing.Brush HorizontalStripe(int n, int p,
  System.Drawing.Color colorBackground, System.Drawing.Color colorLine)
```

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

```
static public System.Drawing.Brush Image(System.Drawing.Image imageIcon)
```

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

```
static public System.Drawing.Brush VerticalStripe(int n, int p,
  System.Drawing.Color colorBackground, System.Drawing.Color colorLine)
```

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.

Description

All of the `Brush` objects returned by the methods in this class are serializable.

Draw Class

Summary

Renders the chart tree to the screen.

`public class Imsl.Chart2D.Draw`

Properties

ClipBounds

`virtual public System.Drawing.Rectangle ClipBounds {get; set; }`

Description

Contains the rectangle to be used for cliping.

DeviceMarkerSize

`virtual public float DeviceMarkerSize {get; }`

Description

The marker size in device coordinates.

Node

`virtual public Imsl.Chart2D.ChartNode Node {set; }`

Description

Specifies a `ChartNode` as the current node.

This is used to get drawing attributes from the tree.

ScaleFont

`virtual public double ScaleFont {get; set; }`

Description

The factor by which fonts are to be scaled.

Constructor

Draw

`public Draw(System.Drawing.Graphics graphics, System.Drawing.Size bounds)`

Description

Contructs 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`.

This gradient is acyclic.

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 x-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.

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.

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.

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.

DrawErrorBar

```
virtual public void DrawErrorBar(int x0, int y0, int x1, int y1, int flag)
```

Description

Draws an error bar.

Legal values: 0=none, 1=bottom, 2=top, 3=both

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

```
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()`

Description

Finish drawing an error bar.

EndFill

`virtual public void EndFill()`

Description

Finish drawing a filled region.

EndImage

`virtual public void EndImage()`

Description

Finish drawing an image.

EndLine

`virtual public void EndLine()`

Description

Finish drawing lines.

EndMarker

`virtual public void EndMarker()`

Description

Finish drawing markers.

EndText

`virtual public void EndText()`

Description

Finish drawing text.

FillArc

```
virtual 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.

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.

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.

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 pointsto add to the graphics path.

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.

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

Summary

FrameChart is a Form that contains a chart.

```
public class Imsl.Chart2D.FrameChart :  Form :
System.ComponentModel.IComponent, System.IDisposable,
System.ComponentModel.ISynchronizeInvoke, System.Windows.Forms.IWin32Window,
System.Windows.Forms.IContainerControl
```

Properties

Chart

`virtual public Imsl.Chart2D.Chart Chart {get; set; }`

Description

Specifies the chart to be handled.

Panel

`virtual public Imsl.Chart2D.PanelChart Panel {get; }`

Description

Specifies a Panel that contains the `Chart` 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.

`true` to release both managed and unmanaged resources; `false` to release only unmanaged resources.

Parameter

`disposing` – A `bool`indicating whether to release both managed and unmanaged resources.

PanelChart Class

Summary

A Windows.Forms.Panel that contains a chart.

```
public class Imsl.Chart2D.PanelChart :  Panel :
System.ComponentModel.IComponent, System.IDisposable,
System.ComponentModel.ISynchronizeInvoke, System.Windows.Forms.IWin32Window
```

Property

Chart

`virtual public Imsl.Chart2D.Chart Chart {get; set; }`

Description

Specifies the Chart to be rendered for in this panel.

Constructors

PanelChart

`public PanelChart()`

Description

Creates a new `PanelChart`.

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

`override void OnPaint(System.Windows.Forms.PaintEventArgs painteventargs)`

Description

Calls the UI delegate's `Paint` method, if the UI delegate is non-null.

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 invoker 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.

Parameter

`painteventargs` – The `PaintEventArgs` with the `Graphics` property for painting the chart.

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.

Description

This class causes the contained chart to be redrawn as necessary.

DrawPick Class

Summary

The DrawPick class.

`public class Imsl.Chart2D.DrawPick :  Draw`

Properties

Node

`override public Imsl.Chart2D.ChartNode Node {set; }`

Description

Specifies the current node of the chart tree.

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.

Constructor

DrawPick

```
public DrawPick(System.Windows.Forms.MouseEventArgs mouseEventArgs,
  System.Drawing.Graphics graphics, System.Drawing.Size bounds)
```

Description

Contructs 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

```
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.

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.

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.

DrawErrorBar

```
virtual public void DrawErrorBar(int x0, int y0, int x1, int y1)
```

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.

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

Finsih drawing an error bar.

EndFill

```
override public void EndFill()
```

Description

Finish drawing a filled region.

EndImage

```
override public void EndImage()
```

Description

Finsih 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.

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.

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.

FillPolygon

```
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 pointsto add to the graphics path.

FillPolygon

```
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.

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()`

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.

PickEventArgs Class

Summary

An event that indicates that a chart element has been selected.

`public class Imsl.Chart2D.PickEventArgs : MouseEventArgs`

Property

Node

`virtual public Imsl.Chart2D.ChartNode Node {get; set; }`

Description

The `ChartNode` 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.

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.

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.

Description

Provides data for the `PickPerformed` event.

See Also

(p. ??)

WebChart Class

Summary

A `WebChart` provides a component to use in ASP.NET applications that holds a `Chart` object.

```
public class Imsl.Chart2D.WebChart :  Panel :
System.ComponentModel.IComponent, System.IDisposable,
System.Web.UI.IParserAccessor, System.Web.UI.IDataBindingsAccessor,
System.Web.UI.IAttributeAccessor
```

Property

Chart

```
public Imsl.Chart2D.Chart Chart {get; set; }
```

Description

The `Chart` object associated with this `WebChart`.

Constructor

WebChart

```
public WebChart()
```

Description

Default constructor.

Methods

OnInit

`override void OnInit(System.EventArgs e)`

Description

Initializes the object.

Parameter

`e` – The `EventArgs` object that contains the event data.

Render

`override void Render(System.Web.UI.HtmlTextWriter output)`

Description

Renders the WebChart to the specified HTML writer.

Parameter

`output` – The `HtmlTextWriter` that receives the control content.

DrawMap Class

Summary

Creates an HTML client-side imagemap from a chart tree.

`public class Imsl.Chart2D.DrawMap : Draw`

Properties

Map

`virtual public string Map {get; }`

Description

Returns the body of the HTML imagemap.

Node

`override public Imsl.Chart2D.ChartNode Node {set; }`

Description

Specifies the current node of the chart tree.

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.

Constructor

DrawMap

`public DrawMap(System.Drawing.Graphics graphics, System.Drawing.Size bounds)`

Description

Contructs 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

`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.

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.

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.

DrawErrorBar

`override public void DrawErrorBar(int x0, int y0, int x1, int y1, int flag)`

Description

Draws an error bar.

Legal values: 0=none, 1=bottom, 2=top, 3=both

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).

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

Finsih 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.

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.

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.

FillPolygon

`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 pointsto add to the graphics path.

FillPolygon

`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.

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.

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()
```

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.

Description

Entries in the imagemap correspond to nodes that define the HREF attribute.

BoxPlot Class

Summary

Draws a multiple-group Box plot.

```
public class Imsl.Chart2D.BoxPlot :  Data
```

Fields

BOXPLOT_TYPE_HORIZONTAL

`public int BOXPLOT_TYPE_HORIZONTAL`

Description

Value for attribute "BoxPlotType" indicating that this is a horizontal box plot.

Used in connection with `BoxPlot` nodes.

BOXPLOT_TYPE_VERTICAL

`public int BOXPLOT_TYPE_VERTICAL`

Description

Value for attribute "BoxPlotType" indicating that this is a horizontal box plot.

Used in connection with `BoxPlot` nodes.

Properties

Bodies

`virtual public Imsl.Chart2D.ChartNode Bodies {get; }`

Description

The main body of the `BoxPlot` elements.

BoxPlotType

`virtual public int BoxPlotType {get; set; }`

Description

Specifies the orientation of the `BoxPlot`.

Legal values are
Imsl.Chart2D.BoxPlot.BOXPLOT$_TYPE_VERTICAL(p.885)orImsl.Chart2D.BoxPlot.BOXPLOT_TYPE_HORIZO$

FarMarkers

`virtual public Imsl.Chart2D.ChartNode FarMarkers {get; }`

Description

The far markers of the `BoxPlot` elements.

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.

`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.

ProportionalWidth

`virtual public bool ProportionalWidth {get; set; }`

Description

Specifies whether the box widths are to be proportional.

`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.

Constructors

BoxPlot

`public BoxPlot(Imsl.Chart2D.AxisXY axis, double[] x, double[][] obs)`

Description

Constructs a box plot chart node with specified x values.

The number of rows in *obs* must equal the length of x. The length of each row in *obs* must be at least 4.

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.

BoxPlot

```
public BoxPlot(Imsl.Chart2D.AxisXY axis, double[] x,
  Imsl.Chart2D.BoxPlot.Statistics[] statistics)
```

Description

Constructs a box plot chart node with specified x values.

The number of `BoxPlot.Statistics[]` must equal `x.Length`.

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*.

BoxPlot

`public BoxPlot(Imsl.Chart2D.AxisXY axis, double[][] obs)`

Description

Constructs a box plot chart.

The length of each row in *obs* must be at least 4.

Parameters

`axis` – An `AxisXY` which is the parent of this node.

`obs` – A `double[]` containing the observations.

Methods

GetStatistics

`virtual public Imsl.Chart2D.BoxPlot.Statistics GetStatistics(int iSet)`

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.

GetStatistics

`virtual public Imsl.Chart2D.BoxPlot.Statistics[] GetStatistics()`

Description

Returns statistics for each set of observations.

Returns

A `BoxPlot.Statistics[]` containing the statistics for each set of observations.

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetDataRange

`override public void SetDataRange(double[] range)`

Description

Update the data range.

The entries in *range* are updated to reflect the extent of the data in this node. *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.

Parameter

`range` – A `double[]` which contains the updated range, {xmin,xmax,ymin,ymax}.

SetLabels

`virtual public void SetLabels(string[] labels, int type)`

Description

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", "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$_{T}YPE_{V}ERTICAL(p.885)orImsl.Chart2D.BoxPlot.BOXPLOT_{T}YPE_{H}ORIZO$

Parameters

`labels` – A `String[]` containing the axis labels.

`type` – An `int` which specifies the "BoxPlotType" attribute value.

SetLabels

`virtual public void SetLabels(string[] labels)`

Description

Sets up an axis with labels.

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.

Parameter

`labels` – A `String[]` containing the axis labels.

Description

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 $\pm 1.58\ IRQ/\sqrt{n}$, 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.

Example: Box Plot Chart

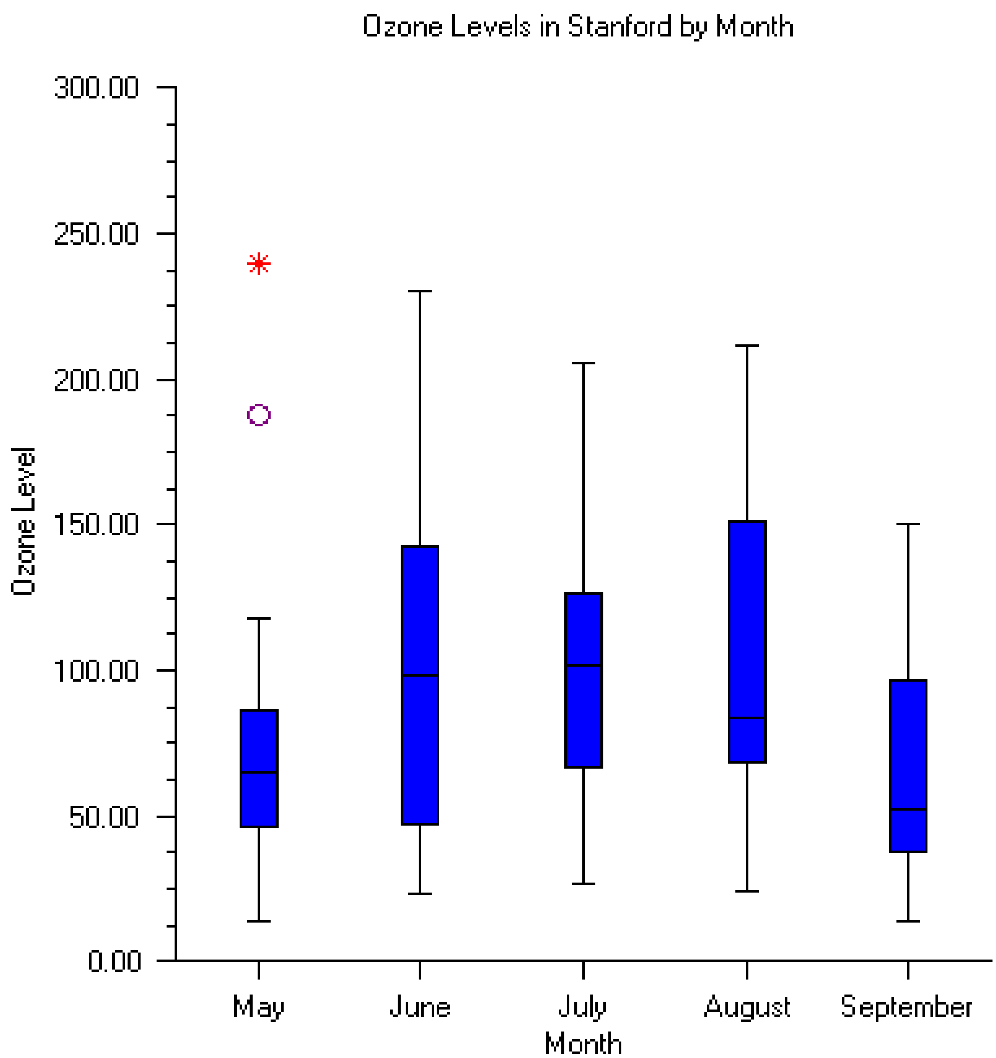
A simple box plot chart is constructed in this example. Display of far and outside values is turned on.

```
using Imsl.Chart2D;
using Imsl.Stat;
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, 174.0, 141.0, 202.0},

                             new double[]{113.0, 38.0, 38.0, 28.0, 52.0, 14.0, 38.0, 94.0,
                                             89.0, 99.0, 150.0, 146.0, 113.0, 38.0, 66.0, 38.0,
                                             80.0, 80.0, 99.0, 71.0, 42.0, 52.0, 33.0, 38.0,
                                             24.0, 61.0, 108.0, 38.0, 28.0}};

        double[] x = new double[]{1.0, 2.0, 3.0, 4.0, 5.0};
        System.String[] xLabels = new System.String[]{"May", "June", "July", "August", "September"};

        //   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());
    }
}
```

Output



Ozone Levels in Stanford by Month
300.00
250.00
200.00
150.00
100.00
50.00
0.00
Ozone Level
May
June
July
August
September
Month

BoxPlot.Statistics Class

Summary

Computes the statistics for one set of observations in a `Boxplot`.

```
public class Imsl.Chart2D.BoxPlot.Statistics
```

Properties

LowerAdjacentValue

`virtual public double LowerAdjacentValue {get; }`

Description

A `double` which contains the lower adjacent value.

LowerQuartile

`virtual public double LowerQuartile {get; }`

Description

A `double` which contains the lower quartile value (25th percentile).

MaximumValue

`virtual public double MaximumValue {get; }`

Description

A `double` which contains the maximum value of this set.

Median

`virtual public double Median {get; }`

Description

A `double` which contains the median value for a set of observations.

MedianLowerConfidenceInterval

`virtual public double MedianLowerConfidenceInterval {get; }`

Description

A `double` which contains the lower confidence interval for the median value of this set of observations.

MedianUpperConfidenceInterval

`virtual public double MedianUpperConfidenceInterval {get; }`

Description

A `double` which contains the upper confidence interval for the median value of this set of observations.

MinimumValue

`virtual public double MinimumValue {get; }`

Description

A `double` which contains the minimum value of this set.

NumberObservations

`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`.

There must be at least 4 observations to compute the statistics.

Parameter

`obs` – A `double[]` containing the set of observations.

`System.ArgumentException` id 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

Summary

A `Contour` chart shows level curves of a two-dimensional function.

```
public class Imsl.Chart2D.Contour :  Data
```

Property

ContourLegend

```
virtual public Imsl.Chart2D.Contour.Legend ContourLegend {get; }
```

Description

Contains the legend information associated with this `Contour`.

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.

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`.

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.

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.

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`.

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.

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.

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. 68)

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.

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.

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. 68)

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.

Methods

GetContourLevel

`virtual public Imsl.Chart2D.ContourLevel GetContourLevel(int k)`

Description

Returns a specified ContourLevel.

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.

Parameter

`k` – An `int` which indicates what `ContourLevel` to return.

Returns

A `ContourLevel` that corrisponds to the k-th level (`cLevel[k]`).

GetContourLevel

`virtual public Imsl.Chart2D.ContourLevel[] GetContourLevel()`

Description

Returns all of the contour levels.

Returns

A `ContourLevel[]` containing the "ContourLevel" attribute value.

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetDataRange

`override public void SetDataRange(double[] range)`

Description

Update the data range.

The entries in *range* are updated to reflect the extent of the data in this node. *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.

Parameter

`range` – A `double[4]` which contains the updated range, {xmin,xmax,ymin, ymax}.

Description

The function can be defined either as values on a rectangular grid or by scattered data points.

A set of ContourLevel (p. 910) objects are created as children of this node. The number of `ContourLevel`s 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. 779), LineWidth (p. 779) and SetMarkerDashPattern (p. 804)) and fill attributes (specified with FillType (p. 796) and FillColor (p. 778))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. 910)

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 two 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

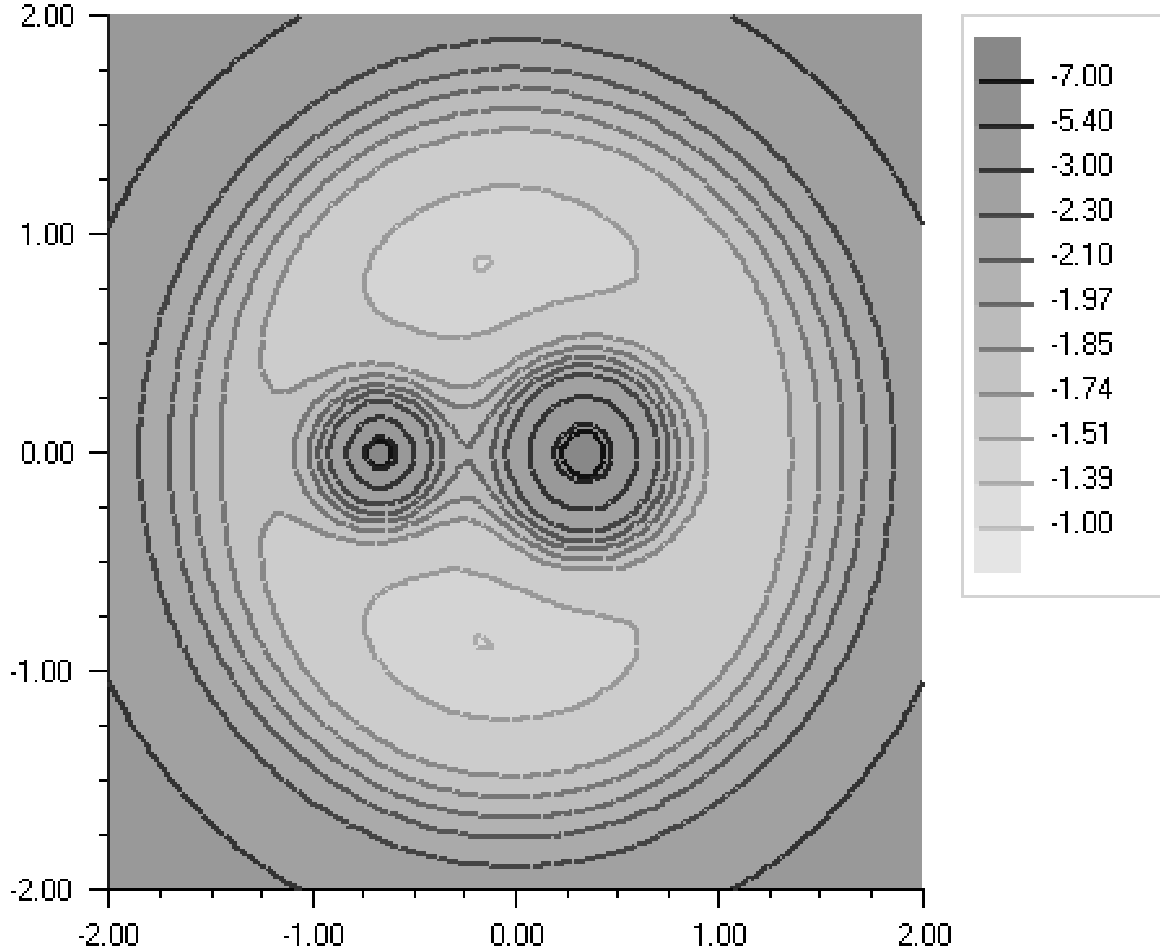
$$V = \frac{m(M_1 + M_2)G}{a} \left[\frac{x_2}{\sqrt{(x - x_1)^2 + y^2}} - \frac{x_1}{\sqrt{(x - x_2)^2 + y^2}} - \frac{1}{2}\left(x^2 + y^2\right) \right]$$

The universal gravitational constant is G. The following program plots the part of $V(x,y)$ inside of the square bracket. The factor $\frac{m(M_1+M_2)G}{a}$ is ignored because it just scales the plot.

```
using Imsl.Chart2D;
using Imsl.Stat;
using System;
using System.Windows.Forms;

public 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);
        }

        // 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 = System.Math.Sqrt((x - x1) * (x - x1) + y * y);
                double d2 = System.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());
    }
}
```

Output


2.00
1.00
0.00
-1.00
-2.00
-2.00
-1.00
0.00
1.00
2.00
-7.00
-5.40
-3.00
-2.30
-2.10
-1.97
-1.85
-1.74
-1.51
-1.39
-1.00

Example: Contour Chart from Scattered Data

In this example, a contour chart is created from 150, randomly choosen, 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.

```
using Imsl.Chart2D;
using Imsl.Stat;
using System;
using System.Windows.Forms;

public 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]=0.7180829622748057;
        randomValue[9]=0.6153607537410654;
        randomValue[10]=0.3158193853638753;
        randomValue[11]=0.10778543304578747;
        randomValue[12]=0.09275375134615693;
        randomValue[13]=0.9817642781628322;
        randomValue[14]=0.467363186309925;
        randomValue[15]=0.9066980293517674;
        randomValue[16]=0.31440695305815347;
        randomValue[17]=0.9991560762956562;
        randomValue[18]=0.785150345014761;
        randomValue[19]=0.7930129038729785;
        randomValue[20]=0.5695413465811706;
        randomValue[21]=0.7625752595574732;
        randomValue[22]=0.0482465474704169;
        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]=0.3204335652731257;
randomValue[36]=0.3501592792324697;
randomValue[37]=0.08674811183862785;
randomValue[38]=0.5605305915601296;
randomValue[39]=0.6088802062708134;
randomValue[40]=0.8382035138841133;
randomValue[41]=0.9236987545556213;
randomValue[42]=0.8024356174828979;
randomValue[43]=0.18382779454152387;
randomValue[44]=0.9443198089192774;
randomValue[45]=0.07466011736504485;
randomValue[46]=0.2961809553169247;
randomValue[47]=0.597869137157411;
randomValue[48]=0.3126393883707773;
randomValue[49]=0.9461805842458413;
randomValue[50]=0.4952325691501952;
randomValue[51]=0.0974865497453884;
randomValue[52]=0.39893060081096055;
randomValue[53]=0.31595422264648054;
randomValue[54]=0.9215776190059227;
randomValue[55]=0.963602405500786;
randomValue[56]=0.1962353914644036;
randomValue[57]=0.897888992070645;
randomValue[58]=0.9816014888911522;
randomValue[59]=0.2591728892012697;
randomValue[60]=0.177119526412298;
randomValue[61]=0.6364841570839579;
randomValue[62]=0.9770940229311096;
randomValue[63]=0.44085669522358406;
randomValue[64]=0.22206796609570068;
randomValue[65]=0.8125478558454153;
randomValue[66]=0.7059166517811799;
randomValue[67]=0.5417895331224579;
randomValue[68]=0.5535562377071471;
randomValue[69]=0.2922863750389211;
randomValue[70]=0.2968612011640126;
randomValue[71]=0.882495829596943;
randomValue[72]=0.9453297028667043;
randomValue[73]=0.5017962685731009;
randomValue[74]=0.17323198276725293;
randomValue[75]=0.516968989592425;
randomValue[76]=0.7264211901923515;
randomValue[77]=0.9589904164393783;
randomValue[78]=0.2896822052185578;
randomValue[79]=0.8709512849886136;
```

```
randomValue[80]=0.3494389711171513;
randomValue[81]=0.444989615581906;
randomValue[82]=0.03683604460307233;
randomValue[83]=0.2794447857758138;
randomValue[84]=0.5426558540369049;
randomValue[85]=0.14701055330017276;
randomValue[86]=0.45822765810918564;
randomValue[87]=0.3804843649168811;
randomValue[88]=0.31543075674256227;
randomValue[89]=0.35478179229078655;
randomValue[90]=0.6740882045962612;
randomValue[91]=0.5722042439512296;
randomValue[92]=0.336494210223919;
randomValue[93]=0.5425187147067986;
randomValue[94]=0.6565124760451249;
randomValue[95]=0.9902292520993252;
randomValue[96]=0.4546287589180955;
randomValue[97]=0.9184888233730713;
randomValue[98]=0.7505359876181693;
randomValue[99]=0.7124220647583559;
randomValue[100]=0.3812755838294607;
randomValue[101]=0.7741986381086996;
randomValue[102]=0.5856540334323093;
randomValue[103]=0.1480175568946106;
randomValue[104]=0.8045988425857213;
randomValue[105]=0.21523348843743784;
randomValue[106]=0.2723138761466122;
randomValue[107]=0.8181756787842892;
randomValue[108]=0.45453852386561255;
randomValue[109]=0.10578123947146922;
randomValue[110]=0.027911361401003143;
randomValue[111]=0.9849840119600158;
randomValue[112]=0.8883835561320729;
randomValue[113]=0.30887148321746527;
randomValue[114]=0.6268231326584466;
randomValue[115]=0.8359413755618763;
randomValue[116]=0.01639605006272593;
randomValue[117]=0.5543612693431772;
randomValue[118]=0.3190057747399081;
randomValue[119]=0.18095345468573598;
randomValue[120]=0.6370180793354232;
randomValue[121]=0.5166986319820245;
randomValue[122]=0.11169309885740164;
randomValue[123]=0.8688720220933366;
randomValue[124]=0.5011922442391221;
randomValue[125]=0.9344952771865647;
randomValue[126]=0.5587227111699117;
randomValue[127]=0.3806089260426023;
randomValue[128]=0.6753272961079825;
randomValue[129]=0.8539394715414731;
randomValue[130]=0.4520234874494251;
randomValue[131]=0.3058558270067878;
randomValue[132]=0.2224399403890832;
randomValue[133]=0.3280806679102708;
randomValue[134]=0.05979465629761105;
randomValue[135]=0.660441325427476;
```

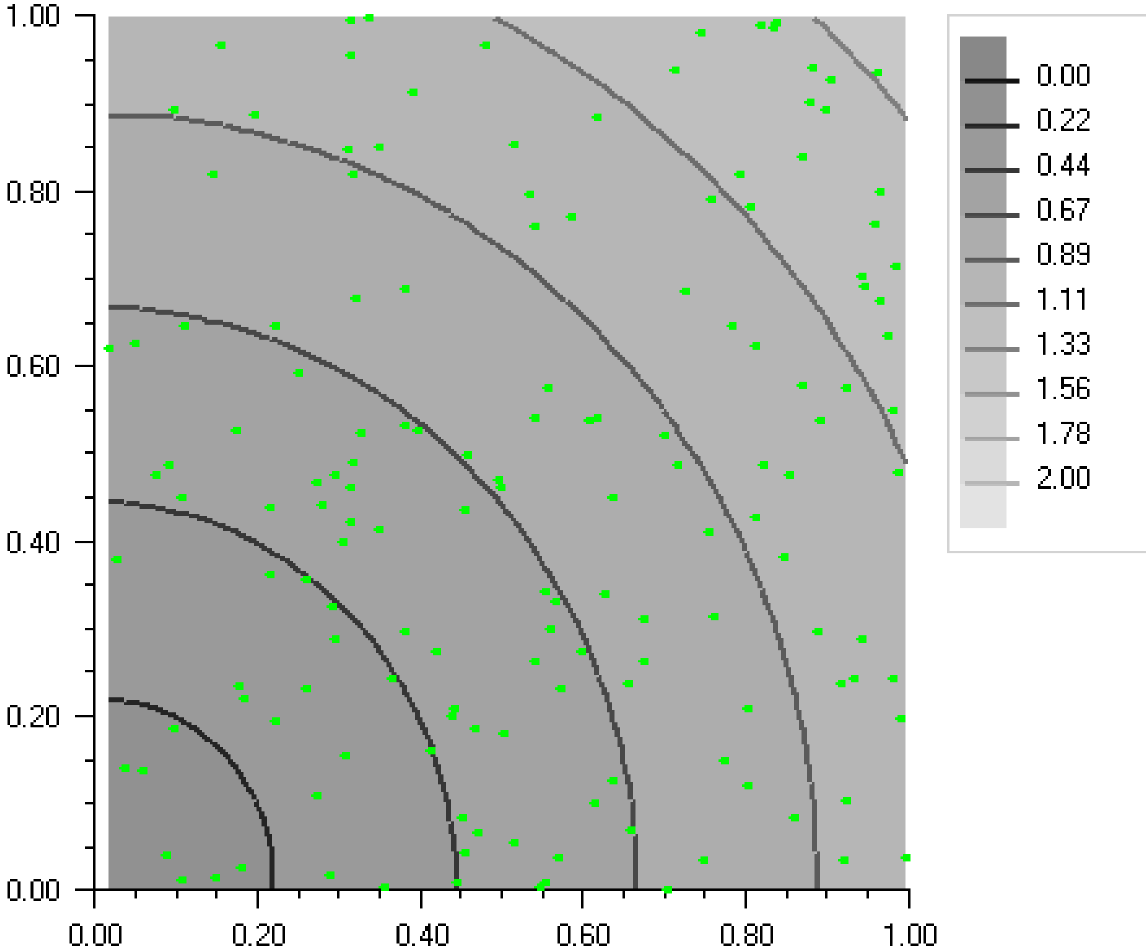
```
randomValue[136]=0.4710041931991943;
randomValue[137]=0.15401687157352573;
randomValue[138]=0.8059082103579294;
randomValue[139]=0.25135648562180013;
randomValue[140]=0.3910396401490016;
randomValue[141]=0.48001615607289505;
randomValue[142]=0.5350655938328643;
randomValue[143]=0.5464799882069644;
randomValue[144]=0.8469694582001581;
randomValue[145]=0.3646033096669923;
randomValue[146]=0.7582401994865531;
randomValue[147]=0.7560344451536601;
randomValue[148]=0.7467799442143332;
randomValue[149]=0.619643401693058;

double[] randomValueY=new double[150];
randomValueY[0]=0.15995876895053263;
randomValueY[1]=0.48794367683379836;
randomValueY[2]=0.20896329070872555;
randomValueY[3]=0.4781765623804778;
randomValueY[4]=0.6732389937186418;
randomValueY[5]=0.33081942994459734;
randomValueY[6]=0.10880787186704965;
randomValueY[7]=0.901138442534768;
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]=0.18652444274911184;
randomValueY[15]=0.9272326533193322;
randomValueY[16]=0.4215880116306273;
randomValueY[17]=0.0386317648903991;
randomValueY[18]=0.6451521871931544;
randomValueY[19]=0.819301055474355;
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]=0.5393314259940907;
randomValueY[30]=0.10341603798162413;
randomValueY[31]=0.48822476962455685;
randomValueY[32]=0.5414223626279245;
randomValueY[33]=0.08241640235000847;
randomValueY[34]=0.27287579633296155;
randomValueY[35]=0.6770605504344167;
randomValueY[36]=0.8497059767892107;
randomValueY[37]=0.04142051621448373;
randomValueY[38]=0.30060172837976995;
randomValueY[39]=0.5378809821731352;
```

```
randomValueY[40]=0.9933333184285308;
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]=0.18480344365963364;
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;
randomValueY[59]=0.354788700886031;
randomValueY[60]=0.2354154511947073;
randomValueY[61]=0.1269624995880959;
randomValueY[62]=0.6337231423679252;
randomValueY[63]=0.19984371337284335;
randomValueY[64]=0.19334220894181153;
randomValueY[65]=0.42648351165619114;
randomValueY[66]=0.0020349209904862997;
randomValueY[67]=0.26227419862014245;
randomValueY[68]=0.010157565396595736;
randomValueY[69]=0.32466354319724255;
randomValueY[70]=0.2880125699286028;
randomValueY[71]=0.942360375989513;
randomValueY[72]=0.28692884801712293;
randomValueY[73]=0.18075667041036092;
randomValueY[74]=0.526829825487406;
randomValueY[75]=0.05392345053644676;
randomValueY[76]=0.6848072074260566;
randomValueY[77]=0.7634213162987096;
randomValueY[78]=0.017226310006998813;
randomValueY[79]=0.8402985996291047;
randomValueY[80]=0.41214609100356114;
randomValueY[81]=0.00903342798862894;
randomValueY[82]=0.13934521987605275;
randomValueY[83]=0.44080857560050446;
randomValueY[84]=0.5420034416544178;
randomValueY[85]=0.8183907621649894;
randomValueY[86]=0.49709491461841304;
randomValueY[87]=0.2960190585426765;
randomValueY[88]=0.4608082576003252;
randomValueY[89]=0.005089578506740633;
randomValueY[90]=0.3108158643301907;
randomValueY[91]=0.23005689707662969;
randomValueY[92]=0.9989728680293828;
randomValueY[93]=0.7588548659179764;
randomValueY[94]=0.23603371611553747;
randomValueY[95]=0.1982727511862804;
```

```
randomValueY[96]=0.04423243217165507;
randomValueY[97]=0.23710549829602878;
randomValueY[98]=0.03408034658051773;
randomValueY[99]=0.9385290439821878;
randomValueY[100]=0.6884926962578499;
randomValueY[101]=0.14803546698365633;
randomValueY[102]=0.7703636833850115;
randomValueY[103]=0.01439471413150828;
randomValueY[104]=0.2089671359503994;
randomValueY[105]=0.4384925493939328;
randomValueY[106]=0.466067663723164;
randomValueY[107]=0.9885280557996187;
randomValueY[108]=0.4343852116079696;
randomValueY[109]=0.4499354044927121;
randomValueY[110]=0.3790637460316687;
randomValueY[111]=0.7145286684532488;
randomValueY[112]=0.2970523498826292;
randomValueY[113]=0.15575074519991794;
randomValueY[114]=0.33981500752026883;
randomValueY[115]=0.9855399747339232;
randomValueY[116]=0.621543401362443;
randomValueY[117]=0.3432116007462742;
randomValueY[118]=0.8180541618673799;
randomValueY[119]=0.027883366004455068;
randomValueY[120]=0.45081070184878236;
randomValueY[121]=0.8533577155496994;
randomValueY[122]=0.6460168649513455;
randomValueY[123]=0.5780055157336823;
randomValueY[124]=0.46048777917596295;
randomValueY[125]=0.24207983525545718;
randomValueY[126]=0.574011233178295;
randomValueY[127]=0.5310197638599929;
randomValueY[128]=0.2621701535374652;
randomValueY[129]=0.4756887402397726;
randomValueY[130]=0.08410532225672551;
randomValueY[131]=0.3991230601447665;
randomValueY[132]=0.6464545787001537;
randomValueY[133]=0.524250367439074;
randomValueY[134]=0.13771323020945658;
randomValueY[135]=0.06816969003124507;
randomValueY[136]=0.06651758347488423;
randomValueY[137]=0.965968335289986;
randomValueY[138]=0.7828616693306287;
randomValueY[139]=0.5906828761391884;
randomValueY[140]=0.9130151004091689;
randomValueY[141]=0.9658950710812012;
randomValueY[142]=0.7969176634278117;
randomValueY[143]=0.003585724779986199;
randomValueY[144]=0.38108388460809595;
randomValueY[145]=0.24225280334829336;
randomValueY[146]=0.7905591927051523;
randomValueY[147]=0.4089325882708409;
randomValueY[148]=0.9802263978904657;
randomValueY[149]=0.8836456558655017;

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


1.00
0.80
0.60
0.40
0.20
0.00
0.00
0.20
0.40
0.60
0.80
1.00
0.00
0.22
0.44
0.67
0.89
1.11
1.33
1.56
1.78
2.00

Contour.Legend Class

Summary

A legend for a contour chart.

`public class Imsl.Chart2D.Contour.Legend : AxisXY`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

This legend should be used for contour charts, instead of usual chart legend.

ContourLevel Class

Summary

`ContourLevel` draws a level curve line and the fill area between the level curve and the next smaller level curve.

`public class Imsl.Chart2D.ContourLevel : ChartNode`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`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. 779), LineWidth (p. 779) and SetMarkerDashPattern (p. 804)) and fill attributes (specified with FillType (p. 796) and FillColor (p. 778))in the `ContourLevel` nodes.

See Also

Imsl.Chart2D.Contour (p. 895)

ErrorBar Class

Summary

Renders data points with error bars.

`public class Imsl.Chart2D.ErrorBar : Data`

Fields

`DATA_TYPE_ERROR_X`
`public int DATA_TYPE_ERROR_X`

Description

Value for attribute "DataType" indicating that this is a horizontal error bar.

Used in connection with `ErrorBar` nodes.

`DATA_TYPE_ERROR_Y`
`public int DATA_TYPE_ERROR_Y`

Description

Value for attribute "DataType" indicating that this is a vertical error bar.

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 `low[k]`,`high[k]`.

If DataType (p. 795) has the bit
Imsl.Chart2D.ErrorBar.DATA$_{T}YPE_{E}RROR_{X}(p.911)setthenthisisahorizontalerrorbar.IfthebitImsl.Chart2D.Erro$

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 arguements have an associated attribute. That is, "X", "Y", "Low" and "High".

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.

`low` – A `double[]` which contains the values which define the minimum extent of the error bars.

`high` – A `double[]` which contains the values which define the maximum extent of the error bars.

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.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetDataRange

`override public void SetDataRange(double[] range)`

Description

Update the data range.

The entries in *range* are updated to reflect the extent of the data in this node. *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.

Parameter

`range` – A `double[4]` which contains the updated range, {xmin,xmax,ymin, ymax}.

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.

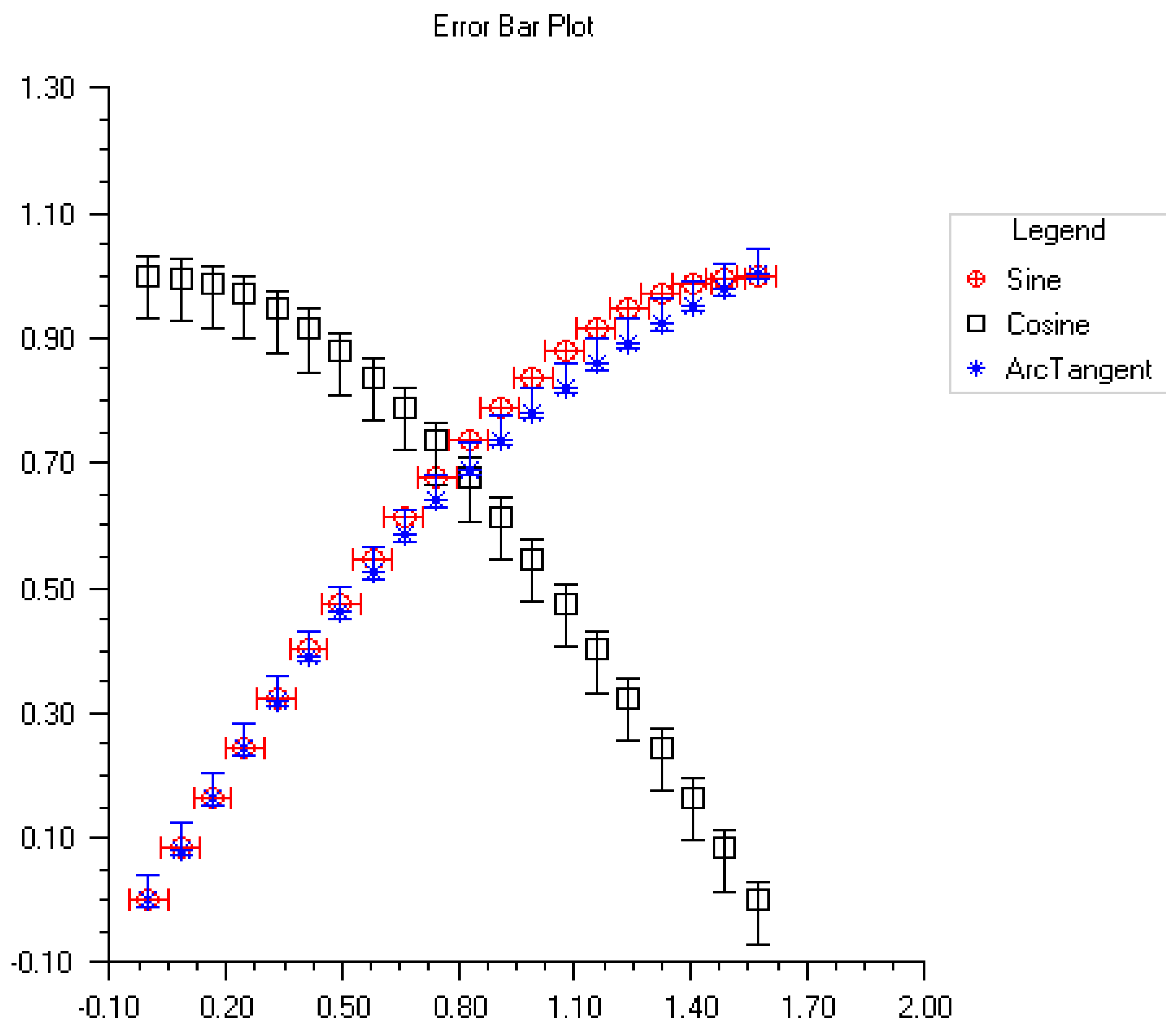
Example: ErrorBar Chart

An ErrorBar chart is constructed in this example. Three data sets are used and a legend is added to the chart. This class can be used either as an applet or as an application.

```
using Imsl.Chart2D;
using Imsl.Stat;
using System;
using System.Windows.Forms;

public 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
        d1.MarkerColor = System.Drawing.Color.Red;
        d2.MarkerColor = System.Drawing.Color.Black;
        d3.MarkerColor = System.Drawing.Color.Blue;

        //  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
        ebar1.MarkerColor = System.Drawing.Color.Red;
        ebar2.MarkerColor = System.Drawing.Color.Black;
        ebar3.MarkerColor = System.Drawing.Color.Blue;

        //  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());
    }
}
```

Output



Error Bar Plot
1.30
1.10
0.90
0.70
0.50
0.30
0.10
-0.10
-0.10
0.20
0.50
0.80
1.10
1.40
1.70
2.00
Legend
Sine
Cosine
ArcTangent

HighLowClose Class

Summary

High-low-close plot of stock data.

`public class Imsl.Chart2D.HighLowClose : 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.

The *high*, *low* and *close* are used to specify the respective attributes. That is, "high", "low" and "close".

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.

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.

The *high*, *low*, *close* and *open* are used to specify the respective attributes. That is, "high", "low", "close" and "open".

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.

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.

The *X*, *high*, *low* and *close* are used to specify the respective attributes. That is, "X", "high", "low" and "close".

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.

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.

The *X*, *high*, *low* and *close* and *open* are used to specify the respective attributes. That is, "X", "high", "low", "close" and "open".

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.

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.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

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.

The entries in *range* are updated to reflect the extent of the data in this node. *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.

Parameter

`range` – A `double[4]` which contains the updated range, {xmin,xmax,ymin, ymax}.

SetDateAxis

`virtual public void SetDateAxis(string labelFormat)`

Description

Sets up the x-axis for high-low-close plot.

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. 780) and TextFormatProvider (p. 781) in the Imsl.Chart2D.AxisLabel (p. 821) node.

Parameter

`labelFormat` – A `string` used to format the date axis labels.

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.

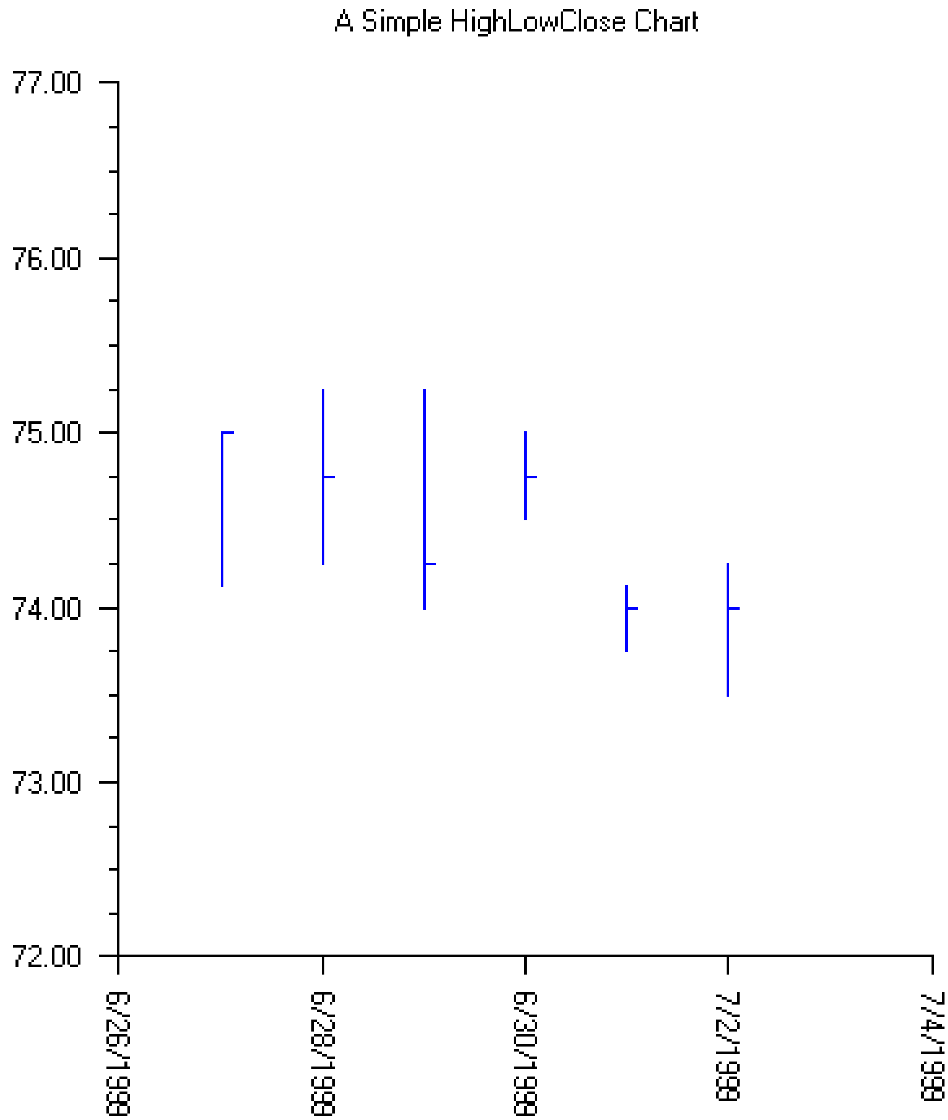
Example: High-Low-Close Chart

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.

```
using Imsl.Chart2D;
using Imsl.Stat;
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);
        hilo.MarkerColor = System.Drawing.Color.Blue;

        //  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());
    }
}
```

Output


A Simple HighLowClose Chart
77.00
76.00
75.00
74.00
73.00
72.00
6/26/1999
6/28/1999
6/30/1999
7/2/1999
7/4/1999

Candlestick Class

Summary

Candlestick plot of stock data.

```
public class Imsl.Chart2D.Candlestick :  HighLowClose
```

Properties

Down

```
virtual public Imsl.Chart2D.CandlestickItem Down {get; }
```

Description

The down days of this `Candlestick`.

Up

```
virtual public Imsl.Chart2D.CandlestickItem Up {get; }
```

Description

The up days of this `Candlestick`.

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.

Each of the arguments are use to set the related attribute (e.g. "High", "Low", "Close" and "Open").

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.

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.

Each of the arguments are use to set the related attribute (e.g. "X", "High", "Low", "Close" and "Open").

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.

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

Two nodes are created as children of this node. One for the up days and one for the down days.

CandlestickItem Class

Summary

A candlestick for the up days or the down days.

`public class Imsl.Chart2D.CandlestickItem : Data`

Method

Paint

```
override public void Paint(Imsl.Chart2D.Draw draw)
```

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

`CandlestickItems` are created by Candlestick; one for up days and one for down days.

See Also

Imsl.Chart2D.Candlestick (p. 924)

SplineData Class

Summary

A data set created from a `Spline`.

```
public class Imsl.Chart2D.SplineData : Data
```

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.

See Also

Imsl.Math.Spline (p. 43)

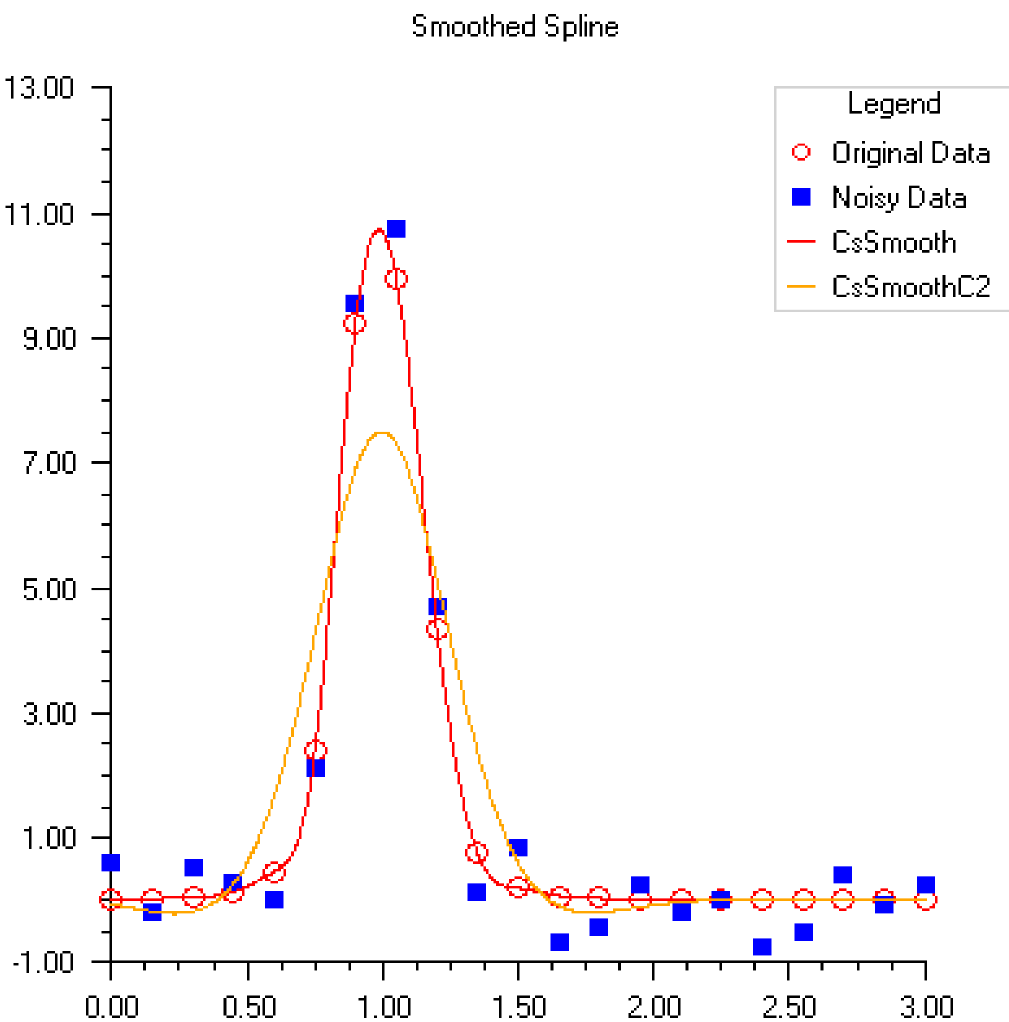
Example: SplineData Chart

This example makes use of the SplineData class as well as the two spline smoothing classes in the package com.imsl.math. This class can be used either as an applet or as an application.

```
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.MarkerColor = System.Drawing.Color.Red;
        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);
        }
        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.MarkerColor = System.Drawing.Color.Blue;
        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");
    }

    static private void chartSpline(AxisXY axis, Imsl.Math.Spline spline, System.Drawing.Color color, System.String
    {
        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());
    }
}
```

Output


Smoothed Spline
Legend
Original Data
Noisy Data
CsSmooth
CsSmoothC2
13.00
11.00
9.00
7.00
5.00
3.00
1.00
-1.00
0.00
0.50
1.00
1.50
2.00
2.50
3.00

Bar Class

Summary

A bar chart.

```
public class Imsl.Chart2D.Bar : Data
```

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 teh 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.

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.

Returns

A `double[][][]` that contains the "BarData" attribute value.

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.

GetBarSet

`virtual public Imsl.Chart2D.BarSet GetBarSet(int group)`

Description

Returns the `BarSet` object.

The group index is assumed to be zero. This method is most useful for charts with only a single group.

Parameter

`group` – An `int` which specifies the group index.

Returns

A `BarSet[][]` containing the "BarSet" attribute value.

GetBarSet

`virtual public Imsl.Chart2D.BarSet[][] GetBarSet()`

Description

Returns the `BarSet` object.

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.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetBarData

`virtual public void SetBarData(double[][][] bardata)`

Description

Sets the "BarData" attribute value.

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.

Parameter

`bardata` – A `double[][][]` that specifies the "BarData" attribute value.

SetDataRange

`override public void SetDataRange(double[] range)`

Description

Update the data range.

The entries in *range* are updated to reflect the extent of the data in this node. *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.

Parameter

`range` – A `double[4]` which contains the updated range, {xmin,xmax,ymin, ymax}.

SetLabels

`virtual public void SetLabels(string[] labels, int type)`

Description

Sets up an axis with bar labels.

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$_TYPE_VERTICAL(p.787)Imsl.Chart2D.ChartNode.BAR_TYPE_HORIZONTAL(p.7$

Parameters

`labels` – A `String[]` which specifes axis labels.

`type` – An `int` which specifies the "BarType".

SetLabels

`virtual public void SetLabels(string[] labels)`

Description

Sets up an axis with bar labels.

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.

Parameter

`labels` – A `String[]` array with which to label the axis.

Description

The class `Bar` has children of class Imsl.Chart2D.BarItem (p. 939). The attribute "BarItem" in class `Bar` is set to the `BarItem` array of children.

See Also

Imsl.Chart2D.BarSet (p. 940), Imsl.Chart2D.BarItem (p. 939)

Example: Stacked Bar Chart

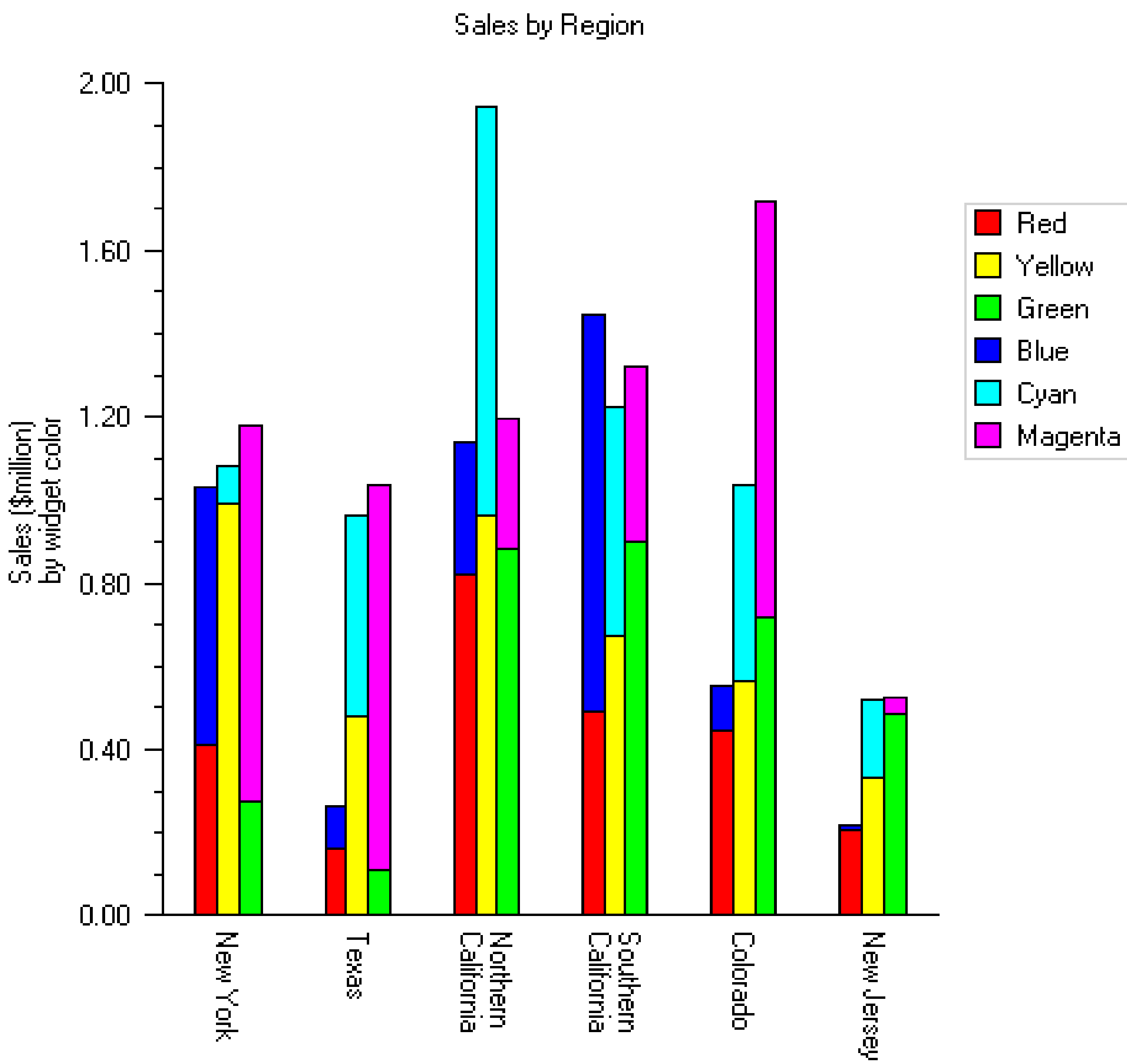
A stacked bar chart is constructed in this example. Bar labels and colors are set and axis labels are set. This class can be used either as an applet or as an application.

```
using Imsl.Chart2D;
using Imsl.Stat;
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
        Imsl.Stat.Random r = new Imsl.Stat.Random(123457);

        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]=0.6732389937186418;
        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]=0.48723656383264413;
        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]=0.9817642781628322;
        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]=0.039285689951912395;
dbl[42]=0.7625752595574732;
dbl[43]=0.31325564481720314;
dbl[44]=0.0482465474704169;
dbl[45]=0.6272275622766595;
dbl[46]=0.09904819350827354;
dbl[47]=0.8934533907186641;
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++)
{
    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;
bar.GetBarSet(0, 1).FillColor = System.Drawing.Color.Yellow;
bar.GetBarSet(0, 2).FillColor = GREEN;
bar.GetBarSet(1, 0).FillColor = System.Drawing.Color.Blue;
```

```
        bar.GetBarSet(1, 1).FillColor = System.Drawing.Color.Cyan;
        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.
        System.String[] labels = new System.String[]{"New York", "Texas", "Northern\nCalifornia", "Southern\nCalifo
        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)\nby " + "widget color");

    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new BarEx1());
    }
}
```

Output



Sales by Region
2.00
1.60
1.20
0.80
0.40
0.00
Sales ($million)
by widget color
Red
Yellow
Green
Blue
Cyan
Magenta
New York
Texas
Northern
California
Southern
California
Colorado
New Jersey

BarItem Class

Summary

A single bar in a bar chart.

`public class Imsl.Chart2D.BarItem :  Data`

Methods

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetDataRange

`override public void SetDataRange(double[] range)`

Description

Update the data range.

The entries in *range* are updated to reflect the extent of the data in this node. *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.

Parameter

`range` – A `double[4]` which contains the updated range, {xmin,xmax,ymin, ymax}.

See Also

Imsl.Chart2D.Bar (p. 930), Imsl.Chart2D.BarSet (p. 940)

BarSet Class

Summary

A set of bars in a bar chart.

```
public class Imsl.Chart2D.BarSet :  ChartNode
```

Methods

GetBarItem

```
virtual public Imsl.Chart2D.BarItem GetBarItem(int index)
```

Description

Returns the `BarItem` given the index.

Parameter

`index` – An `int` which specifies the index.

Returns

A `BarItem` associated with the specified index.

GetBarItem

```
virtual public Imsl.Chart2D.BarItem[] GetBarItem()
```

Description

Returns an array of `BarItems`.

This is the collection of all `BarItems` contained in this bar group.

Returns

A `BarItem[]` that contains the `BarItem` attribute value.

Paint

```
override public void Paint(Imsl.Chart2D.Draw draw)
```

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetDataRange

```
virtual public void SetDataRange(double[] range)
```

Description

Update the data range.

The entries in *range* are updated to reflect the extent of the data in this node. *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.

Parameter

`range` – A `double[4]` which contains the updated range, {xmin,xmax,ymin, ymax}.

Description

A `BarSet`c¿ is created by Imsl.Chart2D.Bar (p. 930) and contains a collection of Imsl.Chart2D.BarItem (p. 939). `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.

Pie Class

Summary

A pie chart.

`public class Imsl.Chart2D.Pie :  Axis`

Constructors

Pie

`public Pie(Imsl.Chart2D.Chart chart)`

Description

Constructs a `Pie` chart object.

The "Viewport" attribute for this node is set to [0.2,0.8] by [0.2,0.8].

Parameter

`chart` – A `Chart` which specifies the parent of this node.

Pie

`public Pie(Imsl.Chart2D.Chart chart, double[] y)`

Description

Constructs a `Pie` chart object with a specified number of slices.

An array of `y.length` Imsl.Chart2D.PieSlice (p. 946) 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].

Parameters

`chart` – A `Chart` which specifies the parent of this node.

`y` – A `double[]` which contains the values for the pie chart.

Methods

GetPieSlice

`virtual public Imsl.Chart2D.PieSlice GetPieSlice(int index)`

Description

Returns a specified `PieSlice`.

The "PieSlice" attribute is a 0 based index array.

Parameter

`index` – An `int` specifying the pie slice to return.

Returns

A `PieSlice` which contains the specified slice.

GetPieSlice

`virtual public Imsl.Chart2D.PieSlice[] GetPieSlice()`

Description

Returns the `PieSlice` objects.

Returns

A `PieSlice[]` containing the pie slices to be associated with this node.

MapDeviceToUser

`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

```
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.

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.

Parameter

`y` – A `double[]` which contains the values for the pie chart.

Returns

A `PieSlice[]` array containing the updated PieSlice.

SetUpMapping

```
override public void SetUpMapping()
```

Description

Initializes the mappings between user and coordinate space.

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.

Description

The angle of the first slice is determined by the attribute ”Reference”.

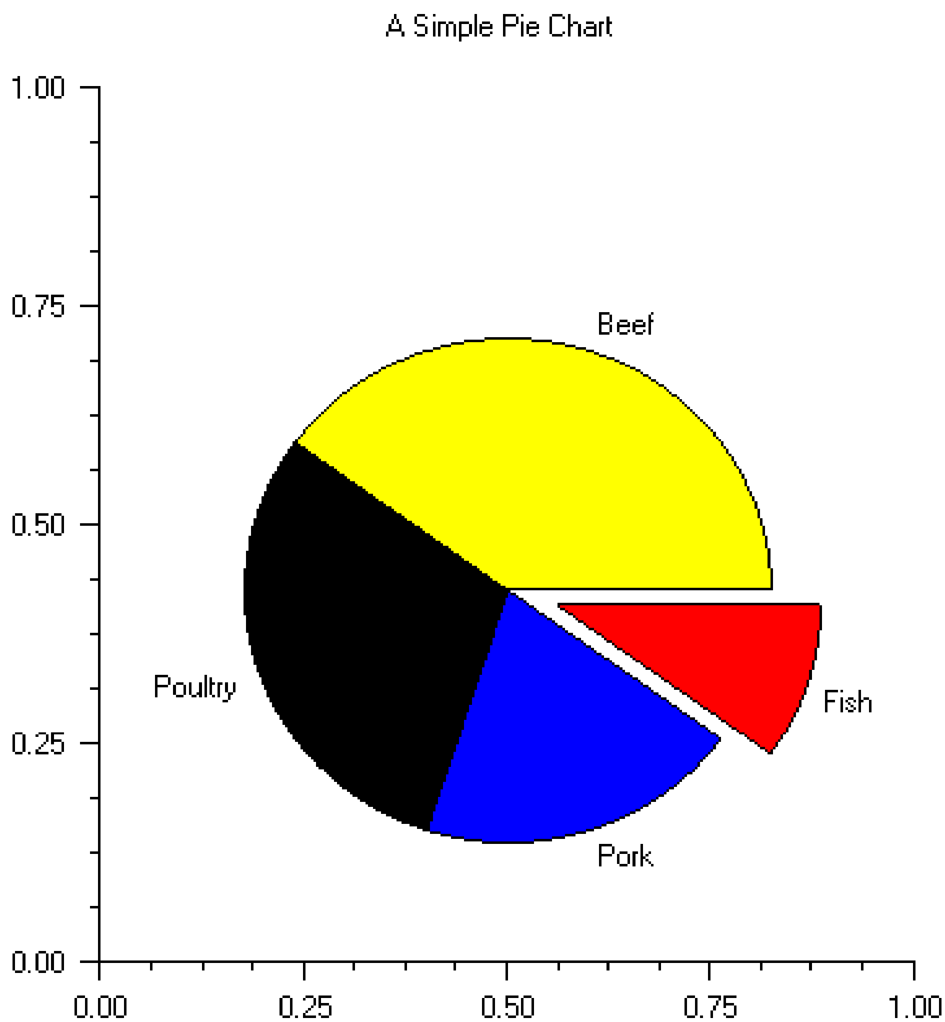
`Pie` is derived from `Axis`, because it defines its own mapping to device space.

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 JFrameChart, which manages the window.

```
using Imsl.Chart2D;
using Imsl.Stat;
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;
        slice[1].FillColor = System.Drawing.Color.Blue;
        slice[2].FillColor = System.Drawing.Color.Black;
        slice[3].FillColor = System.Drawing.Color.Yellow;

        //  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());
    }
}
```

Output


A Simple Pie Chart
1.00
0.75
0.50
0.25
0.00
Beef
Poultry
Fish
Pork
0.00
0.25
0.50
0.75
1.00

PieSlice Class

Summary

One wedge of a pie chart.

`public class Imsl.Chart2D.PieSlice : Data`

Methods

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetAngles

`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.

Description

Imsl.Chart2D.Pie (p. 941) 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. 942). All of the slices can be retrieved using the method Imsl.Chart2D.Pie.GetPieSlice (p. 942).

The drawing of the slice is controlled by the fill attributes (specified with FillType (p. 796)) in this node.

Dendrogram Class

Summary

A `Dendrogram` chart for cluster analysis.

`public class Imsl.Chart2D.Dendrogram : Data`

Properties

Coordinates

`virtual public double[][] Coordinates {get; set; }`

Description

The cluster coordinates in the `Dendrogram` object.

LeftSons

`virtual public int[] LeftSons {get; set; }`

Description

The left sons of each merged cluster.

Levels

`virtual public double[] Levels {get; set; }`

Description

Specifies the levels at which the clusters are joined.

Order

`virtual public int[] Order {get; set; }`

Description

The cluster order in the `Dendrogram` object.

RightSons

`virtual public int[] RightSons {get; set; }`

Description

The right sons of each merged cluster.

Constructors

Dendrogram

```
public Dendrogram(Imsl.Chart2D.AxisXY axis, Imsl.Stat.ClusterHierarchical
  clusterHierarchical)
```

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`.

Dendrogram

```
public 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.

Dendrogram

```
public Dendrogram(Imsl.Chart2D.AxisXY axis, Imsl.Stat.ClusterHierarchical
  clusterHierarchical, int type)
```

Description

Constructs a `Dendrogram` chart using a supplied `ClusterHierarchical` object.

The types possible types of `Dendrogram`s are DENDROGRAM_TYPE_VERTICAL (p. 788) and DENDROGRAM_TYPE_HORIZONTAL (p. 788).

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.

Dendrogram

`public Dendrogram(Imsl.Chart2D.AxisXY axis, double[] clusterLevel, int[] leftSons, int[] rightSons, int type)`

Description

Constructs a `Dendrogram` chart using supplied data.

The types possible types of `Dendrogram`s are DENDROGRAM_TYPE_VERTICAL (p. 788) and DENDROGRAM_TYPE_HORIZONTAL (p. 788).

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.

Methods

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetDataRange

`override public void SetDataRange(double[] range)`

Description

Update the data range.

Parameter

`range` – A `double[4]` which contains the updated range, {xmin,xmax,ymin, ymax}.

SetLabels

`virtual public void SetLabels(string[] labels)`

Description

Sets up the axis labels for `Dendrogram` plot.

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.

Parameter

`labels` – A `String[]` containing the axis labels.

SetLineColors

`virtual public void SetLineColors(System.Drawing.Color[] colors)`

Description

Define colors for individual clusters.

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*.

Parameter

`colors` – A `Color[]` which contains each color to use for the subclusters.

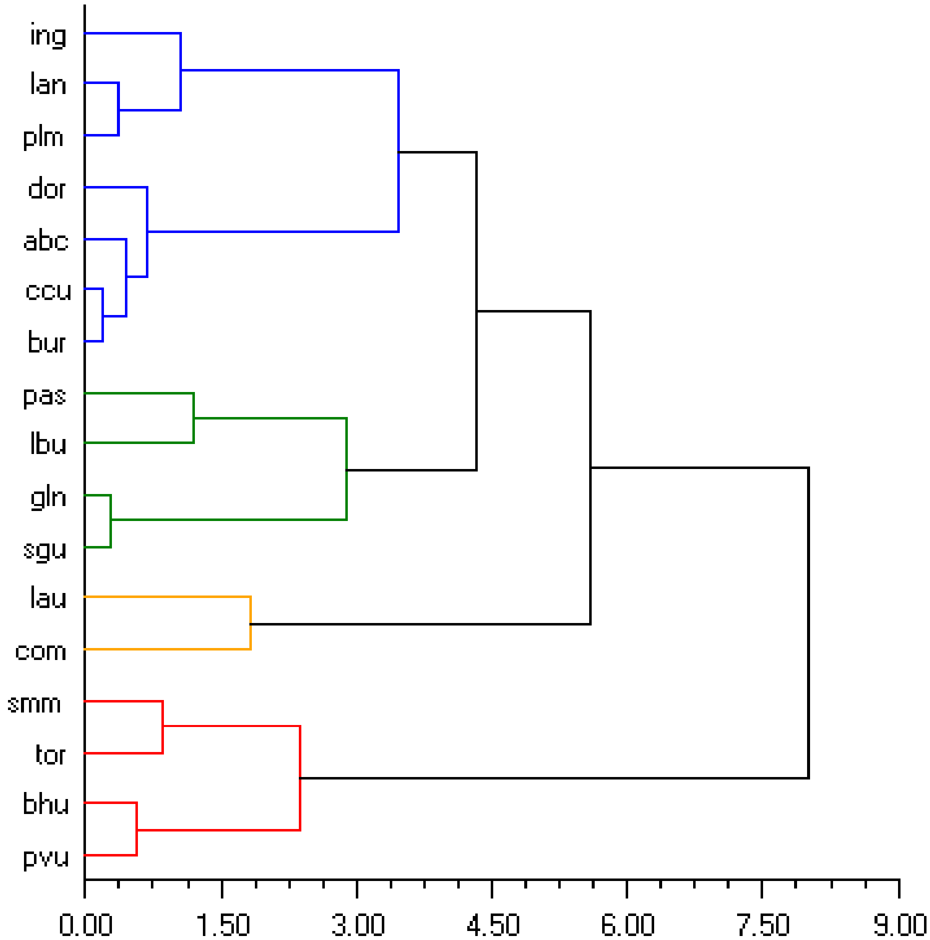
Example: Dendrogram

A Dendrogram.

```
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},
                        {.09, 641.1, 628.8, 629.4},
                        {.12, 638.0, 627.7, 628.6},
                        {.11, 661.4, 659.0, 651.8},
                        {.22, 646.4, 646.2, 647.0},
                        {.33, 634.1, 632.0, 627.8}};

        System.String[] lab = new System.String[]{"lau", "ccu", "bhu", "ing", "com", "smm", "bur", "gln", "pvu", "s

        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());
    }
}
```

Output



Example: Dendrogram

A Dendrogram.

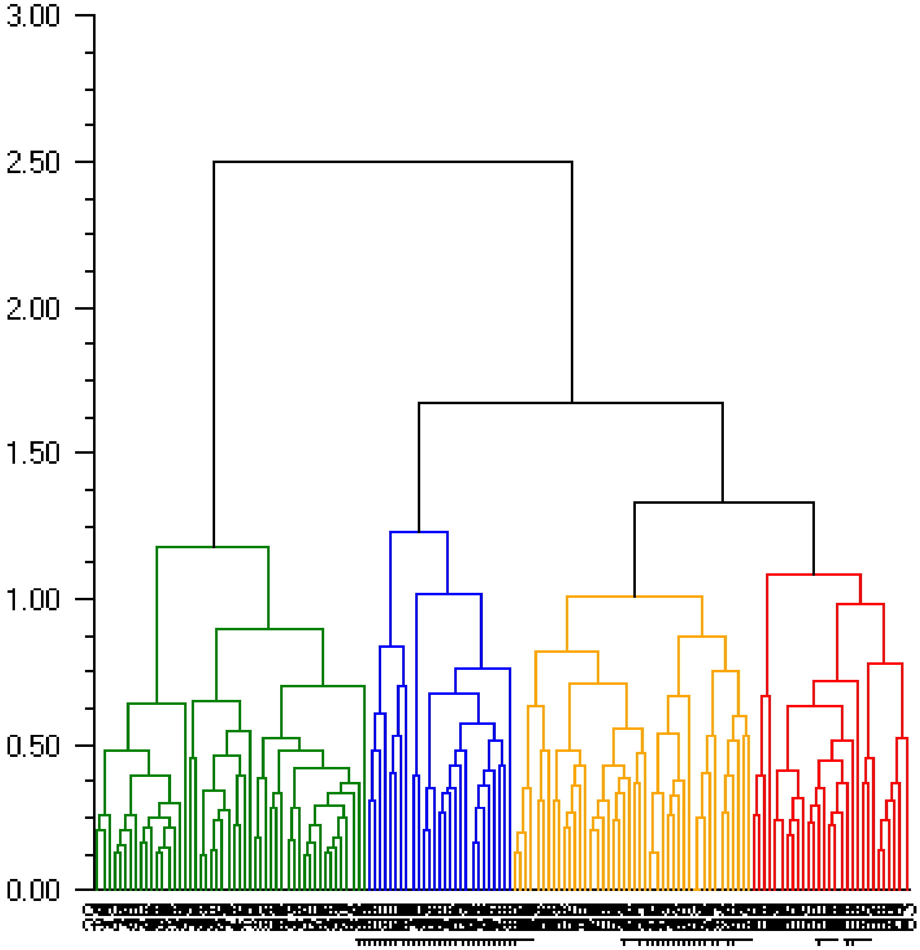
```
using Imsl.Chart2D;
using Imsl.Stat;
using System;
using System.Windows.Forms;
using System.Drawing;

public class DendrogramEx2 : FrameChart
{

    public DendrogramEx2()
    {
        Chart chart = this.Chart;
        AxisXY axis = new AxisXY(chart);

        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},
{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, 2.5},
                        {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(data, 0, 1, 1);
        double[,] distanceMatrix = dist.DistanceMatrix;
        ClusterHierarchical clink = new ClusterHierarchical(dist.DistanceMatrix, 2, 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);
        Dendrogram dc = new Dendrogram(axis, clink.ClusterLevel, clink.ClusterLeftSons, clink.ClusterRightSons);

        // set colors
        dc.SetLineColors(new Color[] {Color.Blue, Color.Green, Color.Red, Color.Orange});

    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new DendrogramEx2());
    }
}
```

Output



Polar Class

Summary

This `Axis` node is used for polar charts.

`public class Imsl.Chart2D.Polar : Axis`

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 specifes 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.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetUpMapping

`override public void SetUpMapping()`

Description

Initializes the mappings between user and coordinate space.

This must be called whenever the screen size, the window or the viewport may have changed.

Description

In a polar plot, the (x,y) coordinates in Imsl.Chart2D.Data (p. 836) nodes are interpreted as (r,theta) values.

Heatmap Class

Summary

`Heatmap` creates a chart from a two-dimensional array of double precision values or Color values.

```
public class Imsl.Chart2D.Heatmap :  Data
```

Properties

Colormap

```
virtual public Imsl.Chart2D.Colormap Colormap {get; set; }
```

Description

Specifies the value of the "Colormap" attribute.

This is the `Colormap` associated with this `Heatmap`. Default: `null`

HeatmapLegend

```
virtual public Imsl.Chart2D.Heatmap.Legend HeatmapLegend {get; }
```

Description

Specifies the heatmap legend.

By default, the legend is not drawn because the IsVisible (p. 779) property is set to `false`. To show the legend set `heatmap.HeatmapLegend.IsVisisble = 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.

The value of *color*`[0,0]` is the color of the cell whose lower left corner is ($xmin$, $ymin$).

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.

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`.

The x-interval (*xmin*, *xmax*) is uniformly divided and mapped into the first index of *data*. The y-interval (*ymin*, *ymax*) 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 (*xmin*, *ymin*).

If a cell has a data value equal to t then its color is the value of the colormap at s, where

$$s = \frac{t - \mathrm{zmin}}{\mathrm{zmax} - \mathrm{zmin}}$$

.

Parameters

`axis` – An `AxisXY` object which specifes 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.

Methods

GetHeatmapLabels

```
virtual public Imsl.Chart2D.Text[,] GetHeatmapLabels()
```

Description

Returns the value of the "HeatmapLabels" attribute.

Default: `null`

Returns

A `Text[,]` that contains the values of the "HeatmapLabels" attribute.

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

SetDataRange

`override public void SetDataRange(double[] range)`

Description

Update the data range.

The entries in *range* are updated to reflect the extent of the data in this node. *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.

Parameter

`range` – A `double[4]` which contains the updated range, {xmin,xmax,ymin, ymax}.

SetHeatmapLabels

`virtual public void SetHeatmapLabels(Imsl.Chart2D.Text[,] labels)`

Description

Sets the value of the "HeatmapLabels" attribute.

The default alignment for `Text` is `TEXT_X_CENTER|TEXT_Y_CENTER`.

See Also: Imsl.Chart2D.Text (p. 850), TEXT_X_CENTER (p. 793), TEXT_Y_CENTER (p. 793)

Parameter

`labels` – A `Text[,]` that specifies the `Heatmap` labels.

SetHeatmapLabels

`virtual public void SetHeatmapLabels(string[,] labels)`

Description

Sets the value of the "HeatmapLabels" attribute.

Each `Text` object is created from the corresponding label value with `TEXT_X_CENTER|TEXT_Y_CENTER` alignment.

See Also: Imsl.Chart2D.Text (p. 850), TEXT_X_CENTER (p. 793), TEXT_Y_CENTER (p. 793)

Parameter

`labels` – A `string[,]` used to create a `Text[,]` that specifies the `Heatmap` labels.

Description

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. 960)

Example: Heatmap from Color array

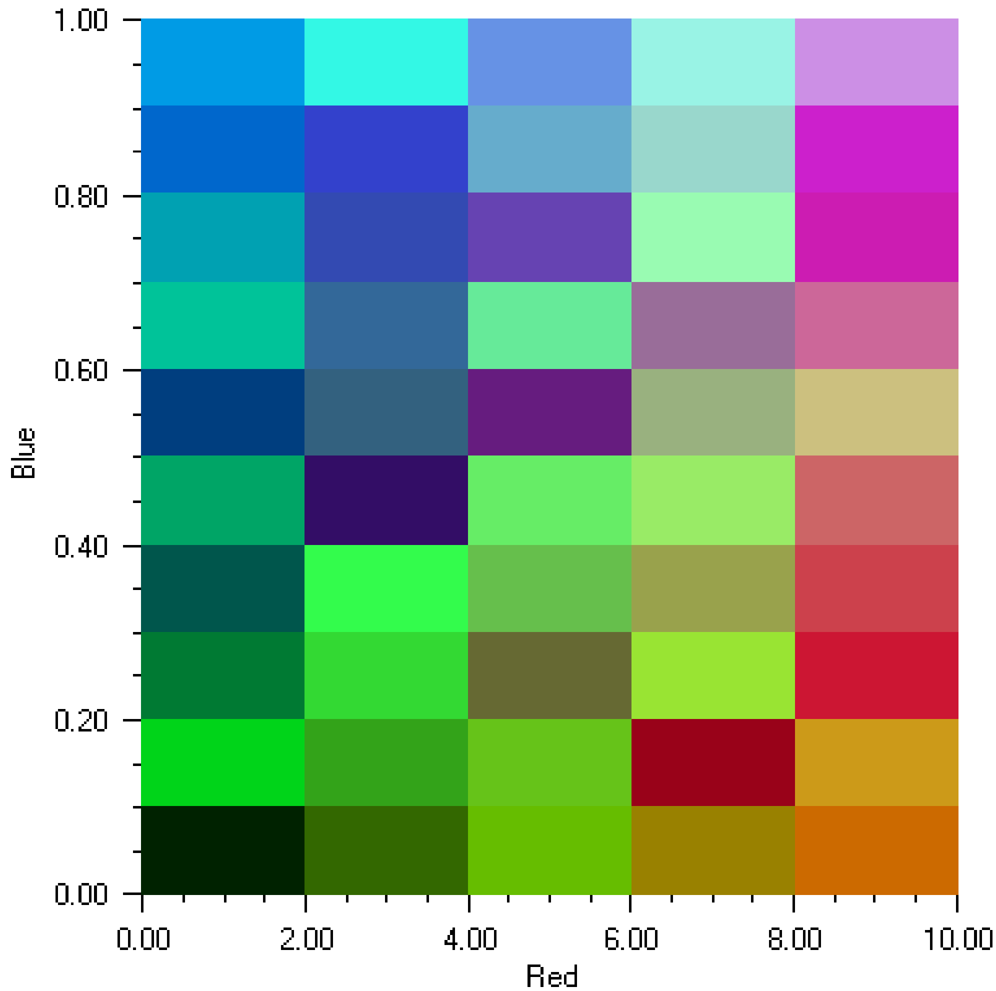
A 5 by 10 array of `Color` objects is created by linearly interpolating red along the x-axis, blue along the y-axis and mixing in a random amount of green. The data range is set to [0,10] by [0,1].

```
using Imsl.Chart2D;
using Imsl.Stat;
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);
System.Drawing.Color[,] color = new System.Drawing.Color[nxRed,nyBlue];

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[11]=163;
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]=235;
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());
    }
}
```

Output



Example: Heatmap from Color array

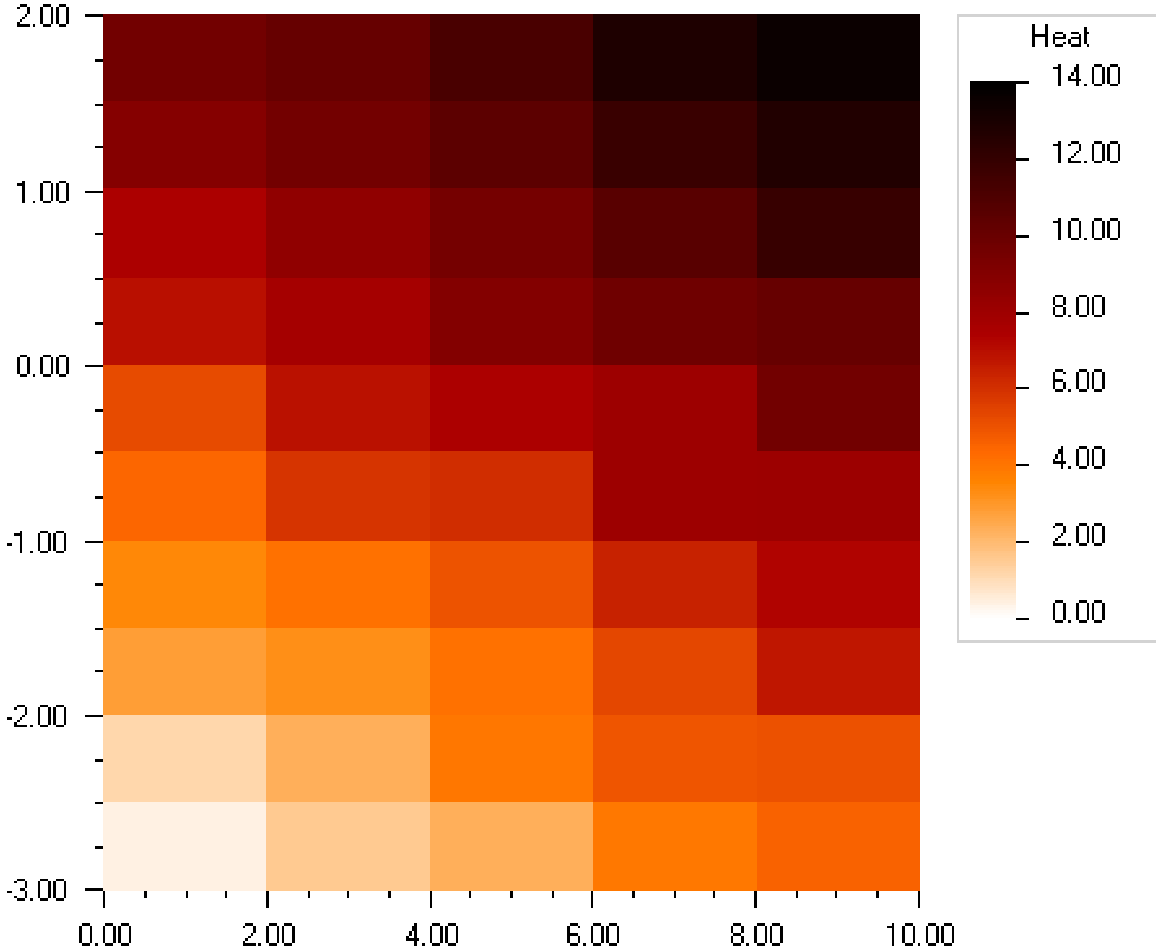
A 5 by 10 data array is created by linearly interpolating from the lower left corner to the upper right corner and adding in a uniform random variable. A red temperature color map is used. This maps the minimum data value to light green and the maximum data value to dark green.

The legend is enabled by setting its paint attribute to true.

```
using Imsl.Chart2D;
using Imsl.Stat;
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 fmin = 0.0;
        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]=0.41312962995625035;
        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]=0.6732389937186418;
        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]=0.48723656383264413;
        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]=0.9817642781628322;
        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]=0.039285689951912395;
        dbl[42]=0.7625752595574732;
        dbl[43]=0.31325564481720314;
        dbl[44]=0.0482465474704169;
        dbl[45]=0.6272275622766595;
        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] = i + j + dbl[z];
                z++;
            }
        }
        Heatmap heatmap = new Heatmap(axis, xmin, xmax, ymin, ymax, 0.0, fmax, data, Imsl.Chart2D.Colormap_Fields.R
        heatmap.HeatmapLegend.IsVisible = true;
        heatmap.HeatmapLegend.SetTitle("Heat");

    }

    public static void Main(string[] argv)
    {
        System.Windows.Forms.Application.Run(new HeatmapEx2());
    }
}
```

Output


2.00
1.00
0.00
-1.00
-2.00
-3.00
0.00
2.00
4.00
6.00
8.00
10.00
Heat
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00

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.

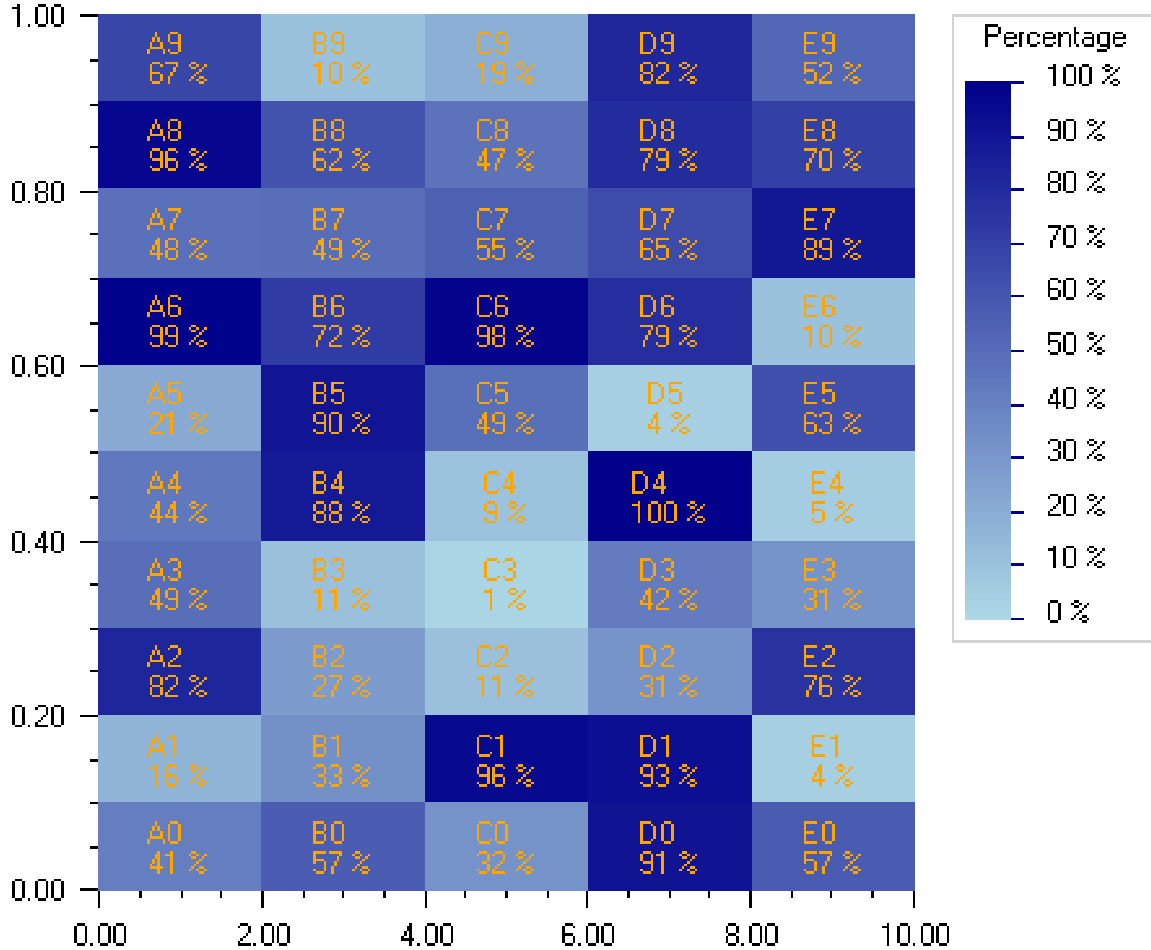
The legend is enabled by setting its paint attribute to true. The tick marks in the legend are formatted using the percentage `NumberFormat` object. A title is also set in the legend.

```
using Imsl.Chart2D;
using Imsl.Stat;
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;

//      SupportClass.TextNumberFormat format = SupportClass.TextNumberFormat.GetTextNumberPercentInstance();

        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]=0.41312962995625035;
        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]=0.6732389937186418;
        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]=0.48723656383264413;
        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]=0.9817642781628322;
        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]=0.039285689951912395;
        dbl[42]=0.7625752595574732;
        dbl[43]=0.31325564481720314;
        dbl[44]=0.0482465474704169;
        dbl[45]=0.6272275622766595;
        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.BL
        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());
```

```
    }
}
```

Output



1.00
0.80
0.60
0.40
0.20
0.00
0.00
2.00
4.00
6.00
8.00
10.00
A9 67 %
B9 10 %
C9 19 %
D9 82 %
E9 52 %
A8 96 %
B8 62 %
C8 47 %
D8 79 %
E8 70 %
A7 48 %
B7 49 %
C7 55 %
D7 65 %
E7 89 %
A6 99 %
B6 72 %
C6 98 %
D6 79 %
E6 10 %
A5 21 %
B5 90 %
C5 49 %
D5 4 %
E5 63 %
A4 44 %
B4 88 %
C4 9 %
D4 100 %
E4 5 %
A3 49 %
B3 11 %
C3 1 %
D3 42 %
E3 31 %
A2 82 %
B2 27 %
C2 11 %
D2 31 %
E2 76 %
A1 16 %
B1 33 %
C1 96 %
D1 93 %
E1 4 %
A0 41 %
B0 57 %
C0 32 %
D0 91 %
E0 57 %
Percentage
100 %
90 %
80 %
70 %
60 %
50 %
40 %
30 %
20 %
10 %
0 %

Heatmap.Legend Class

Summary

A legend for use with a `Heatmap`.

`public class Imsl.Chart2D.Heatmap.Legend :  AxisXY`

Method

Paint

`override public void Paint(Imsl.Chart2D.Draw draw)`

Description

Paints this node and all of its children.

This is normally called only by the `Paint` method in this node's parent.

Parameter

`draw` – A `Draw` which is to be painted.

Description

This `Legend` should be used with `Heatmap`s, rather than the usual `Chart` legend.

Colormap Interface

Summary

`public interface Imsl.Chart2D.Colormap`

Method

GetColor

`abstract public System.Drawing.Color GetColor(double t)`

Description

Maps the parameterization interval [0,1] into `Color`s.

Parameter

`t` – A `double` in the interval [0,1] to be mapped.

Returns

A `Color` corrisponding to *t*.

Colormap_Fields Structure

Summary

`Colormaps` are mappings from the unit interval to `Colors`.

`public structure Imsl.Chart2D.Colormap_Fields`

Fields

`BLUE`
`public Imsl.Chart2D.Colormap BLUE`

Description

A linear blue colormap.

`BLUE_GREEN_RED_YELLOW`
`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.

WHITE_BLUE_LINEAR
`public Imsl.Chart2D.Colormap WHITE_BLUE_LINEAR`

Description

A linear white and blue colormap.

Description

They are a one-dimensional parameterized path through the color cube.

See Also

Imsl.Chart2D.Heatmap (p. 960)

Chapter 24: Neural Nets

Types

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.

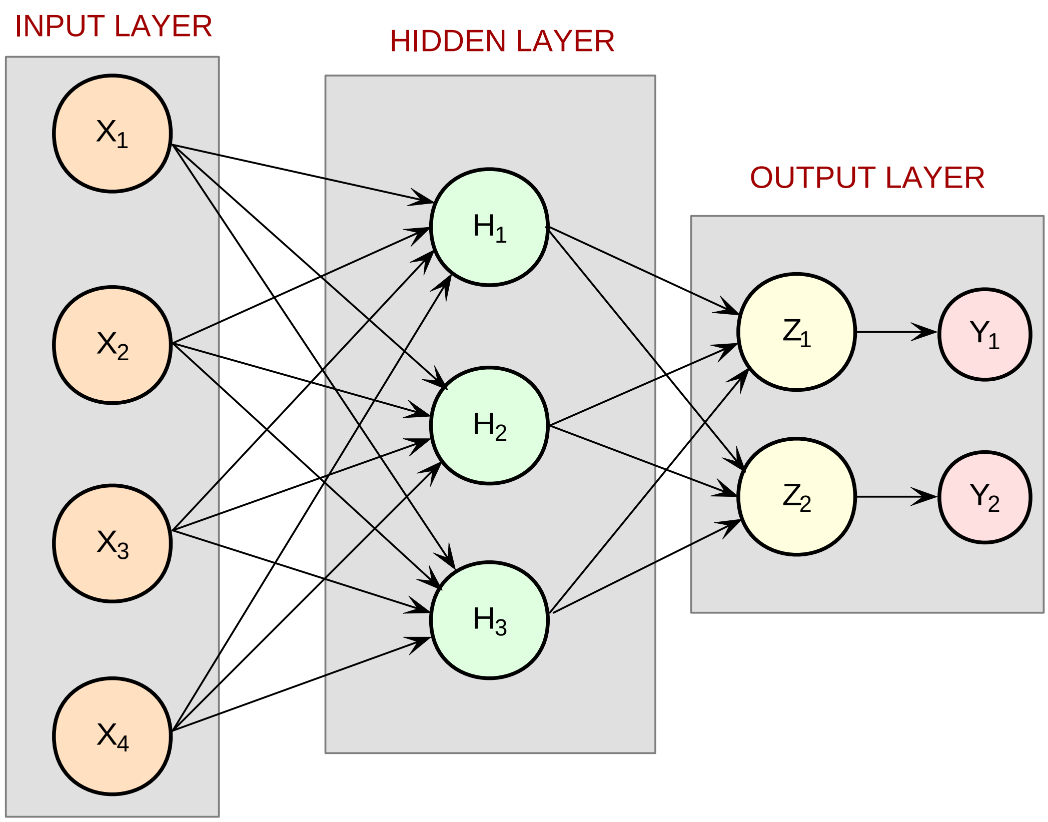


Figure 1. A 2-layer, Feed-Forward Network with 4 Inputs and 2 Outputs

Other networks, such as recurrent neural networks, allow data and information to flow in both directions, see Mandic and Chambers (2001).

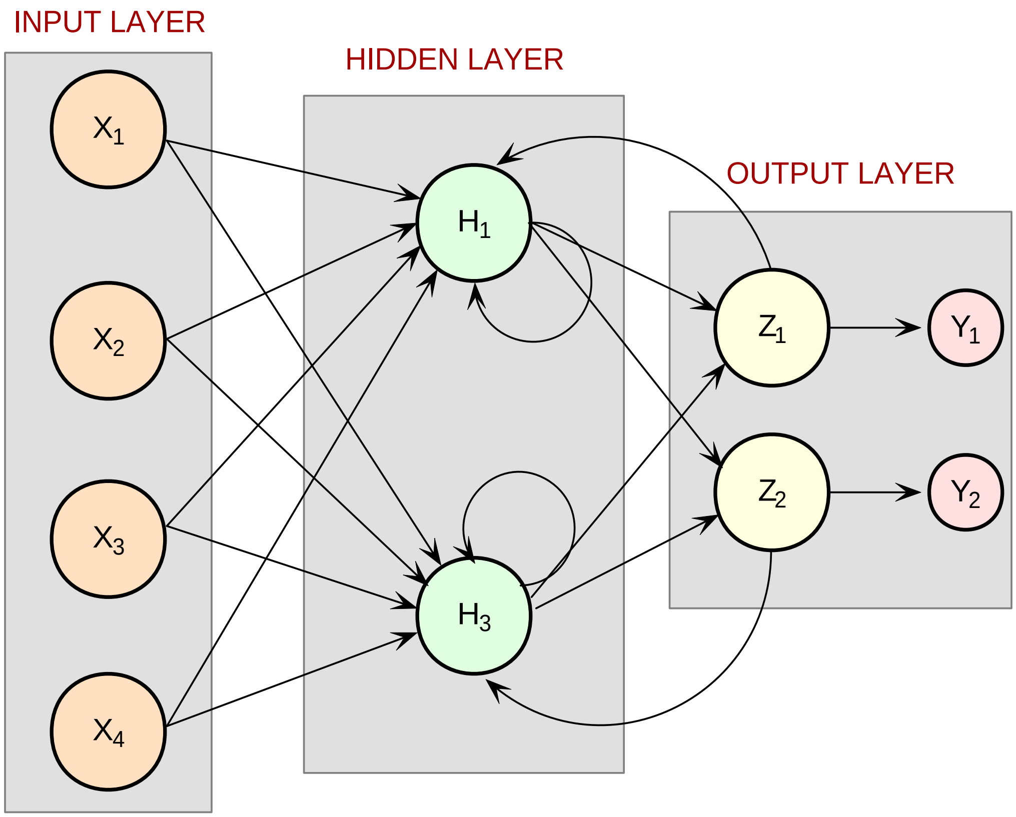


Figure 2. A Recurrent Neural Network with 4 Inputs and 2 Outputs

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	Traditional Statistical Terminology	Description
Training	Model Fitting	Estimating unknown parameters or coefficients in the analysis.
Patterns	Cases or Observations	A single observation of all input and output variables.
Attributes	Independent variables	Inputs to the network or model.
Classes	Dependent variables	Outputs from the network or model calculations.

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.

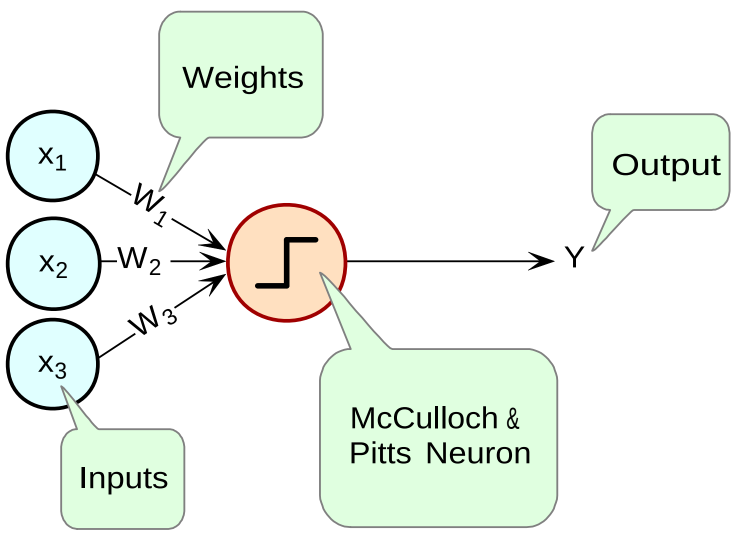


Figure 3. The McCulloch and Pitts Threshold Neuron

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 μ 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, $Z = \sum_{i=1}^{m} w_i x_i - \mu$. 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.

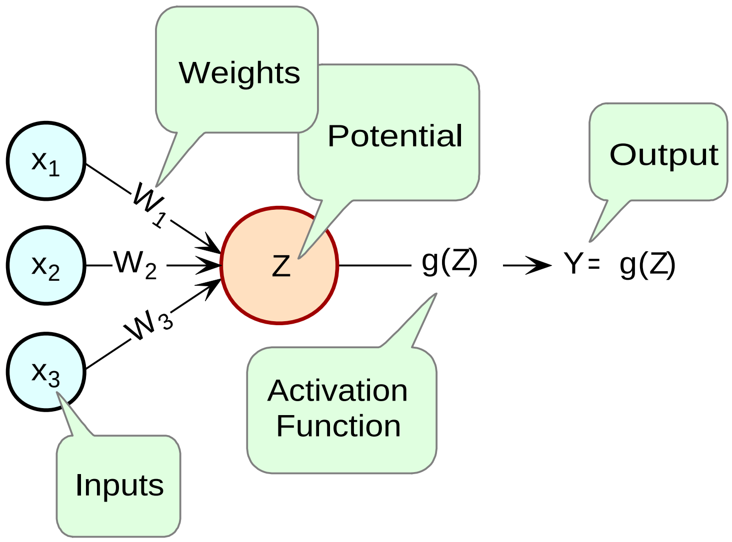


Figure 4. The Perceptron

As shown in Figure 4, perceptrons consist of the following five components:

Component	**Example**
Inputs	X_1,X_2,X_3
Input Weights	W_1,W_2,W_3
Potential	$Z = \sum_{i=1}^{3} W_i X_i - \mu$, where μ is a bias correction.
Activation Function	$g(Z)$
Output	$g(Z)$

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.

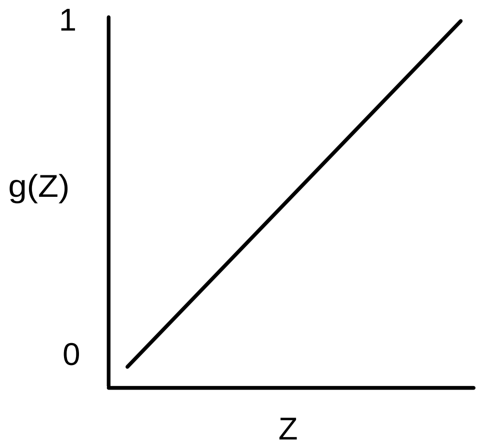


Figure 5. An Identity (Linear) Activation Function

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 $x_1, x_2, \cdots, x_k$ are the k network inputs, Y_i is the i-th network output and $\beta_0, \beta_1, \cdots, \beta_k$ 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., $\mathbb{R}^K \to \mathbb{R}$ where K is the number of perceptron inputs.

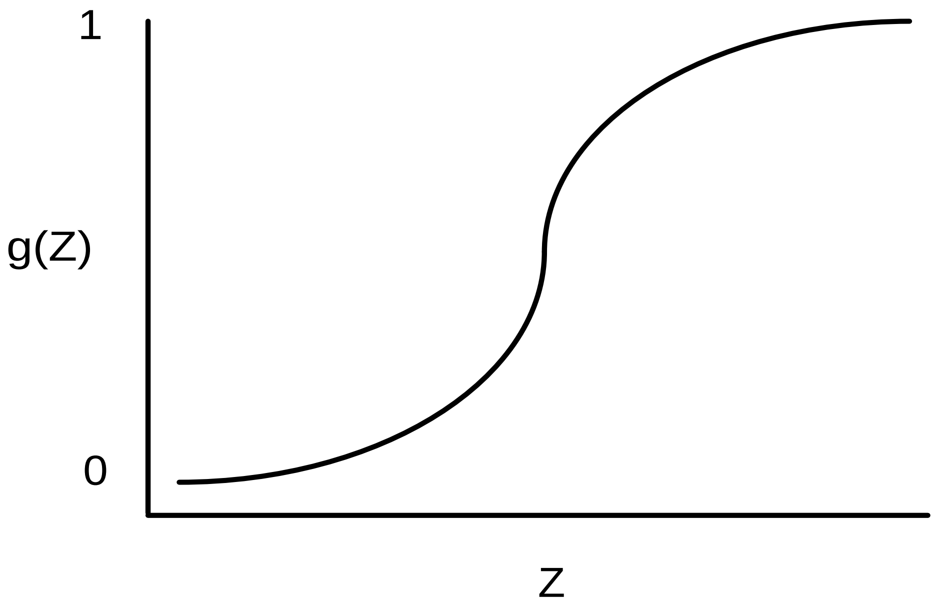


Figure 6. A Sigmoid Activation Function

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 $g(Z) = \tanh(Z) = \frac{e^{\alpha Z} - e^{-\alpha Z}}{e^{\alpha Z} + e^{-\alpha Z}}$
2. the arc-tangent $g(Z) = \frac{2}{\pi} \arctan\left(\frac{\pi Z}{2}\right)$, 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., $\mathbb{R}^M \to \mathbb{R}^C$.

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:

- natural language processing: Manning and Schütze (1999)
- speech and text recognition: Lippmann (1989)
- face recognition: Lawrence, et al. (1997)
- playing backgammon, Tesauro (1990)
- classifying financial news, Calvo (2001).

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),
- and Radial Basis Function Networks, Bishop (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 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 $\mathrm{Z_i}$ 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

$$\mathrm{softmax_i} = \frac{\mathrm{e}^{Z_i}}{\sum_{j=1}^{C} e^{Z_j}}$$

The softmax activation function ensures that the outputs all conform to the requirements for multivariate probabilities. That is,

$$0 < \mathrm{softmax}_i < 1, \quad \text{for all}\, i = 1, 2, \ldots, C$$

and

$$\sum_{i=1}^{C} \mathrm{softmax}_i = 1$$

A pattern is assigned to the i-th classification when $\mathrm{softmax_i}$ 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.

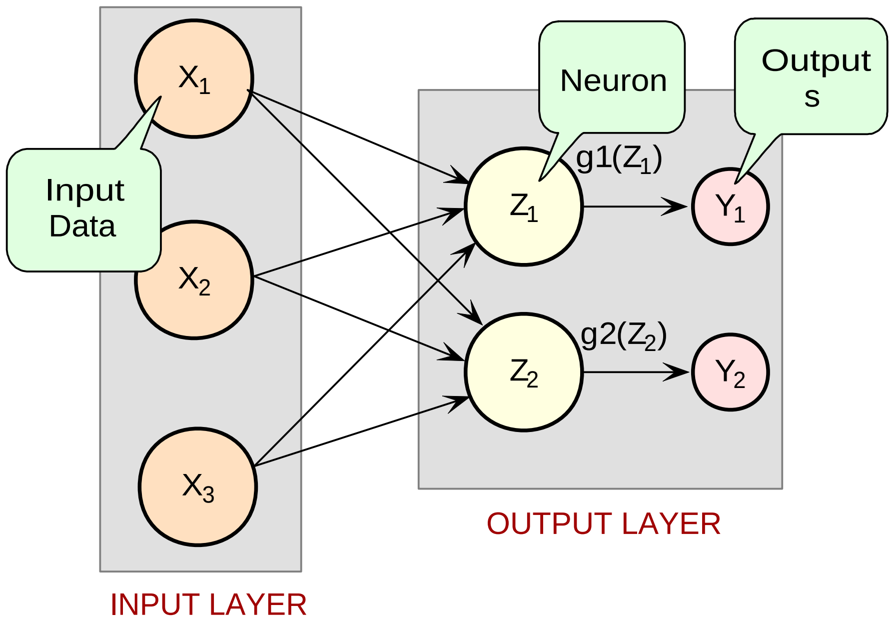


Figure 7. A Single-Layer Feed-Forward Neural Net

In this single-layer feed-forward neural network, the networks 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:

$$E = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{C} \left(t_{ij} - \hat{t}_{ij} \right)^2$$

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=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:

$$E^{RMS} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{C}\left(t_{ij} - \hat{t}_{ij}\right)^2}{\sum_{i=1}^{N}\sum_{j=1}^{C}\left(t_{ij} - \bar{t}\right)^2}$$

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^{RMS} does not increase as N increases. The smaller the value of E^{RMS} the closer the network is predicting its targets during training. A value of $E^{RMS} = 0$ indicates that the network is able to predict every pattern exactly. A value of $E^{RMS} = 1$ 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:

$$E^{RMS} = \frac{2}{\bar{t}} \cdot E$$

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^{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:

$$E^{R} = \sum_{i=1}^{N}\sum_{j=1}^{C}\left|t_{ij} - \hat{t}_{ij}\right|^{R}.$$

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^{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:

$$L = E^1 = \sum_{i=1}^{N}\sum_{j=1}^{C}\left|t_{ij} - \hat{t}_{ij}\right|.$$

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^{RMS}, a scaled version of the Laplacian error can be calculated using the following formula:

$$L^{RMS} = \frac{\sum\limits_{i=1}^{N}\sum\limits_{j=1}^{C}\left|t_{ij} - \hat{t}_{ij}\right|}{\sum\limits_{i=1}^{N}\sum\limits_{j=1}^{C}\left|t_{ij} - \bar{t}\right|}$$

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.:

$$t_{ij} = \begin{cases} 1 & \text{if training pattern } i{=}yes \\ 0 & \text{if the training pattern } i{=}no \end{cases}$$

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):

$$E^C = -\sum_{i=1}^{N} \left\{ t_i \ln(\hat{t}_i) + (1-t_i)\ln(1-\hat{t}_i) \right\}$$

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:

$$E^C_{\min} = -\sum_{i=1}^{N} \{ t_i \ln(t_i) + (1-t_i)\ln(1-t_i) \}$$

Subtracting this from E^C gives an error term bounded below by zero, i.e., $E^{CE} \geq 0$ where:

$$E^{CE} = E^C - E^C_{\min} = -\sum_{i=1}^{N} \left\{ t_i \ln\left[\frac{\hat{t}_i}{t_i}\right] + (1-t_i)\ln\left[\frac{1-\hat{t}_i}{1-t_i}\right] \right\}$$

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:

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{C}\left(t_{ij} - \hat{t}_{ij}\right)^2$$

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 δ_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 2.Activation Functions and Their Derivatives

Activation Function	$g(a)$	$g'(a)$
Linear	$g(a) = a$	$g'(a) = 1$ (where a is a constant)
Logistic	$g(a) = \frac{1}{1+e^{-a}}$	$g'(a) = g(a)(1 - g(a))$
Hyperbolic-tangent	$g(a) = \tanh(a)$	$g'(a) = \text{sech}^2(a) = 1 - \tanh^2(a)$
Squash	$g(a) = \frac{a}{1+\lvert a\rvert}$	$g'(a) = \frac{1}{(1+\lvert a\rvert)^2}$

Creating a Feed Forward Network

The following code fragment creates the feed forward neural network shown in the following figure:

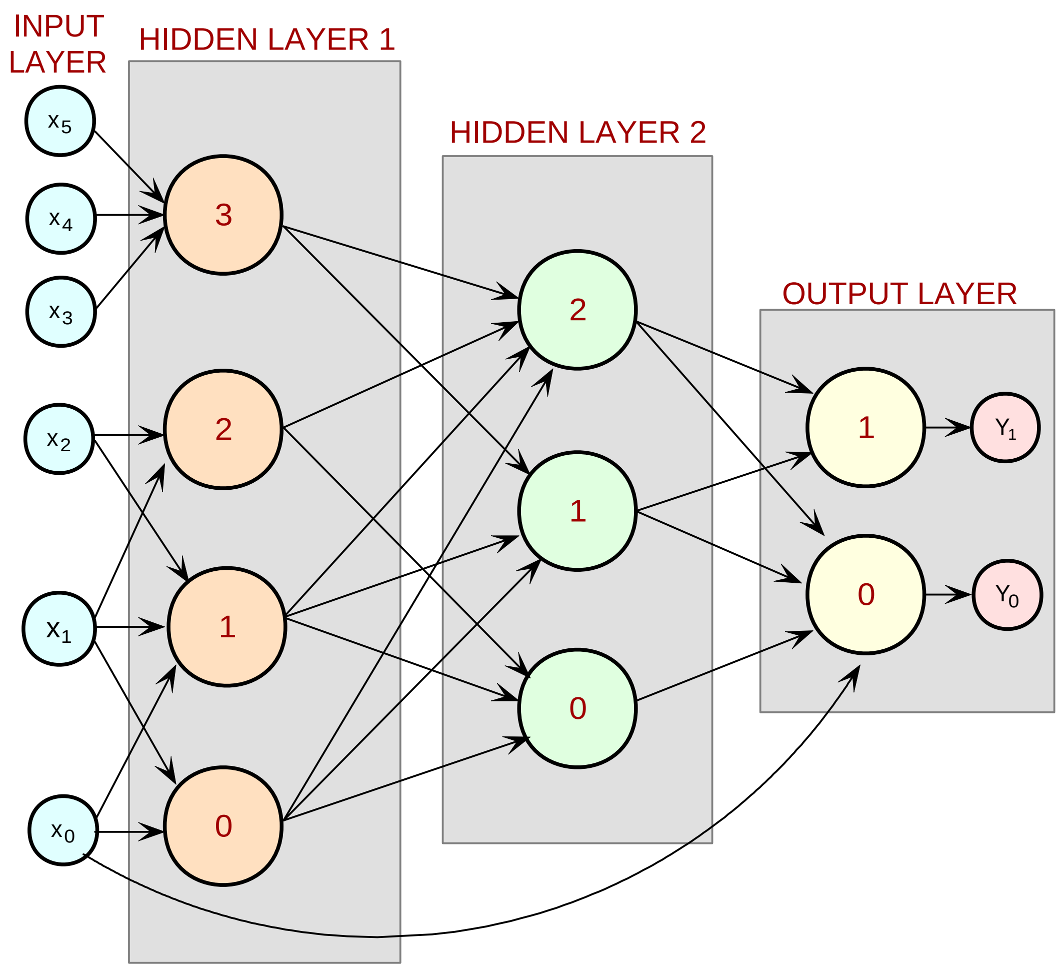


Figure 8. A Three-Layer Feed-Forward Neural Net

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.

```
// ************************************************************
// EXAMPLE CODE FOR CREATING LINKS AMONG NETWORK NODES
// ************************************************************

   FeedForwardNetwork network = new FeedForwardNetwork();
   network.InputLayer.CreateInputs(6);
   network.CreateHiddenLayer().CreatePerceptrons(4);
   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[1], outputNode[1]);
   network.Link(layer2Node[2], outputNode[0]);
   network.Link(layer2Node[2], outputNode[1]);
// Create link between input node[0] and ouput 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[2],hiddenLayer1[3]));
   network.Remove(network.FindLink(inputNode[3],hiddenLayer1[0]));
   network.Remove(network.FindLink(inputNode[3],hiddenLayer1[1]));
   network.Remove(network.FindLink(inputNode[3],hiddenLayer1[2]));
   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[1]));
   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. Fourteen 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
// ***************************************************************
   double xData[,] = ...
   double yData[,] = ...
   QuasiNewtonTrainer trainer = new QuasiNewtonTrainer();
   trainer.GradientTolerance = 1.0e-7;
   trainer.Train(network, xData, yData);
.
.
.
// ***************************************************************
```

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:

```
// ***************************************************************
// 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 consists 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:

```
// ***************************************************************
// 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.

```
// ****************************************************************
// EXAMPLE CODE FOR SAVING TRAINED NETWORK USING SERIALIZATION
// ****************************************************************
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
.
.
.
// ****************************************************************
// 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 targests
   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;
using System.Runtime.Serialization.Formatters.Binary;
.
.
.
// ******************************************************************
// READ THE TRAINED NETWORK FROM THE SERIALIZED NETWORK OBJECT
   Network network = (Network)read("MyNetwork.ser");
// READ THE SERIALIZED 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

Summary

A representation of a feed forward neural network.

`public class Imsl.DataMining.Neural.FeedForwardNetwork : Network`

Properties

HiddenLayers

`virtual public Imsl.DataMining.Neural.HiddenLayer[] HiddenLayers {get; }`

Description

The `HiddenLayers` in this Imsl.DataMining.Neural.Network (p. 1148).

InputLayer

`override public Imsl.DataMining.Neural.InputLayer InputLayer {get; }`

Description

The `InputLayer` in this Imsl.DataMining.Neural.Network (p. 1148).

Links

`override public Imsl.DataMining.Neural.Link[] Links {get; }`

Description

All the `Links` in this Imsl.DataMining.Neural.Network (p. 1148).

NumberOfInputs

`override public int NumberOfInputs {get; }`

Description

The number of inputs to the Imsl.DataMining.Neural.Network (p. 1148).

NumberOfLinks

`override public int NumberOfLinks {get; }`

Description

The number of Links (p. 1007) in the Imsl.DataMining.Neural.Network (p. 1148).

NumberOfOutputs

```
override public int NumberOfOutputs {get; }
```

Description

The number of outputs from the Imsl.DataMining.Neural.Network (p. 1148).

NumberOfWeights

```
override public int NumberOfWeights {get; }
```

Description

The number of Weights (p. 1029) in the Imsl.DataMining.Neural.Network (p. 1148).

OutputLayer

```
override public Imsl.DataMining.Neural.OutputLayer OutputLayer {get; }
```

Description

The neural network `OutputLayer`.

Perceptrons

```
override public Imsl.DataMining.Neural.Perceptron[] Perceptrons {get; }
```

Description

The `Perceptrons` in this Imsl.DataMining.Neural.Network (p. 1148).

Weights

```
override public double[] Weights {get; set; }
```

Description

The Weights (p. 1029) for the Links (p. 1007) in this Imsl.DataMining.Neural.Network (p. 1148).

The array contains the `Weights` for each `Link` followed by the `Perceptron` Imsl.DataMining.Neural.Perceptron.Bias (p. 1026) values. The `Link` `Weights` are the order in which the `Links` were created. The `Weight` values are first, followed by the `Bias` values in the Imsl.DataMining.Neural.HiddenLayer (p. 1020) and then the `Bias` values in the Imsl.DataMining.Neural.FeedForwardNetwork.OutputLayer (p. 1005), and then by the order in which the Perceptrons (p. 1026) were created.

Constructor

FeedForwardNetwork

```
public FeedForwardNetwork()
```

Description

Creates a new instance of `FeedForwardNetwork`.

Methods

CreateHiddenLayer

`override public Imsl.DataMining.Neural.HiddenLayer CreateHiddenLayer()`

Description

Creates a `HiddenLayer`.

Returns

A `HiddenLayer` object which specifies a neural network hidden layer.

FindLink

```
virtual public Imsl.DataMining.Neural.Link
  FindLink(Imsl.DataMining.Neural.Node from, Imsl.DataMining.Neural.Node to)
```

Description

Returns the `Link` between two `Node`s.

Parameters

`from` – The origination `Node`.

`to` – The destination `Node`.

Returns

A `Link` between the two `Node`s, or `null` if no such `Link` exists.

FindLinks

```
virtual public Imsl.DataMining.Neural.Link[]
  FindLinks(Imsl.DataMining.Neural.Node to)
```

Description

Returns all of the `Link`s to a given `Node`.

Parameter

`to` – A `Node` whose `Link`s are to be determined.

Returns

An array of `Link`s containing all of the `Link`s to the given `Node`.

Forecast

`override public double[] Forecast(double[] x)`

Description

Computes a forecast using the Imsl.DataMining.Neural.Network (p. 1148).

Parameter

`x` – A `double` array of values to which the Nodes (p. 1024) in the Imsl.DataMining.Neural.FeedForwardNetwork.InputLayer (p. 1004) are to be set.

Returns

A `double` array containing the values of the `Node`s in the Imsl.DataMining.Neural.FeedForwardNetwork.OutputLayer (p. 1005).

GetForecastGradient

```
override public double[,] GetForecastGradient(double[] xData)
```

Description

Returns the derivatives of the outputs with respect to the Weights (p. 1029).

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.

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.

Link

```
virtual public Imsl.DataMining.Neural.Link Link(Imsl.DataMining.Neural.Node
  from, Imsl.DataMining.Neural.Node to, double weight)
```

Description

Establishes a `Link` between two `Nodes` with a specified Weight (p. 1029).

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`.

Link

```
virtual public Imsl.DataMining.Neural.Link Link(Imsl.DataMining.Neural.Node
  from, Imsl.DataMining.Neural.Node to)
```

Description

Establishes a `Link` between two `Nodes`.

Any existing `Link` between these `Nodes` is removed.

Parameters

`from` – The origination `Node`.

`to` – The destination `Node`.

Returns

A `Link` between the two `Nodes`.

LinkAll

```
virtual public void LinkAll()
```

Description

For each Imsl.DataMining.Neural.Layer (p. 1018) in the Imsl.DataMining.Neural.Network (p. 1148), link each Imsl.DataMining.Neural.Node (p. 1024) in the `Layer` to each `Node` in the next `Layer`.

LinkAll

```
virtual public void LinkAll(Imsl.DataMining.Neural.Layer from,
  Imsl.DataMining.Neural.Layer to)
```

Description

Links all of the Nodes (p. 1024) in one `Layer` to all of the `Nodes` in another `Layer`.

Parameters

`from` – The origination `Layer`.

`to` – The destination `Layer`.

Remove

```
virtual public void Remove(Imsl.DataMining.Neural.Link link)
```

Description

Removes a `Link` from the Imsl.DataMining.Neural.Network (p. 1148).

Parameter

`link` – The `Link` deleted from the `Network`.

SetEqualWeights

```
virtual public void SetEqualWeights(double[,] xData)
```

Description

Initializes network weights using equal weighting.

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 hidden layer if the number of layers is greater than 1 or the output layer, 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 intial values at each training stage. If the epoch trainer is used, noise is added to these initial values at the start of each epoch.

Parameter

`xData` – An input `double` matrix containing training patterns. The number of columns in `xData` must equal the number of nodes in the input layer.

SetRandomWeights

```
virtual public void SetRandomWeights(double[,] xData, System.Random random)
```

Description

Initializes network weights using random weights.

The random weights algorithm assigns equal weights to all perceptrons, except those in the first layer connected to the input layer. Like the equal weights algorithm, perceptrons not in the first layer are assigned weights $1/k$, where k is the number of inputs connected to that perceptron.

For the first layer perceptron weights, they 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 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 intial values at each training stage. If the epoch trainer is used, noise is added to these initial values at the start of each epoch.

Parameters

`xData` – An input `double` matrix containing training patterns. The number of columns in `xData` must equal the number of nodes in the input layer.

`random` – A `Random` object.

ValidateLink

`virtual protected internal void ValidateLink(Imsl.DataMining.Neural.Node from, Imsl.DataMining.Neural.Node to)`

Description

Checks that a Imsl.DataMining.Neural.FeedForwardNetwork.Link(Imsl.DataMining.Neural.Node,Imsl.DataMining.Neural.Node) (p. 1007) between two `Nodes` is valid.

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.

Parameters

`from` – The origination `Node`.

`to` – The destination `Node`.

`System.ArgumentException` id is thrown if the `Link` is not valid

Description

A `Network` contains an Imsl.DataMining.Neural.FeedForwardNetwork.InputLayer (p. 1004), an Imsl.DataMining.Neural.FeedForwardNetwork.OutputLayer (p. 1005) and zero or more HiddenLayers (p. 1020). The `null InputLayer` and `OutputLayer` are automatically created by the `Network` constructor. The InputNodes (p. 1025) are added using the `FeedForwardNetwork.InputLayer.CreateInputs(nInputs)` method. Output Perceptrons (p. 1026) are added using the `FeedForwardNetwork.OutputLayer.CreatePerceptrons(nOutputs)`, and `HiddenLayers` can be created using the `FeedForwardNetwork.CreateHiddenLayer().CreatePerceptrons(nPerceptrons)` method.

The `InputLayer` contains `InputNodes`. The `HiddenLayers` and `OutputLayers` contain `Perceptron` nodes. These Nodes (p. 1024) are created using factory methods in the Layers (p. 1018).

The `Network` also contains Links (p. 1007) between `Nodes`. `Links` are created by methods in this class.

Each `Link` has a Weight (p. 1029) and Gradient value. Each `Perceptron` node has a Bias (p. 1026) 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.

Example: FeedForwardNetwork

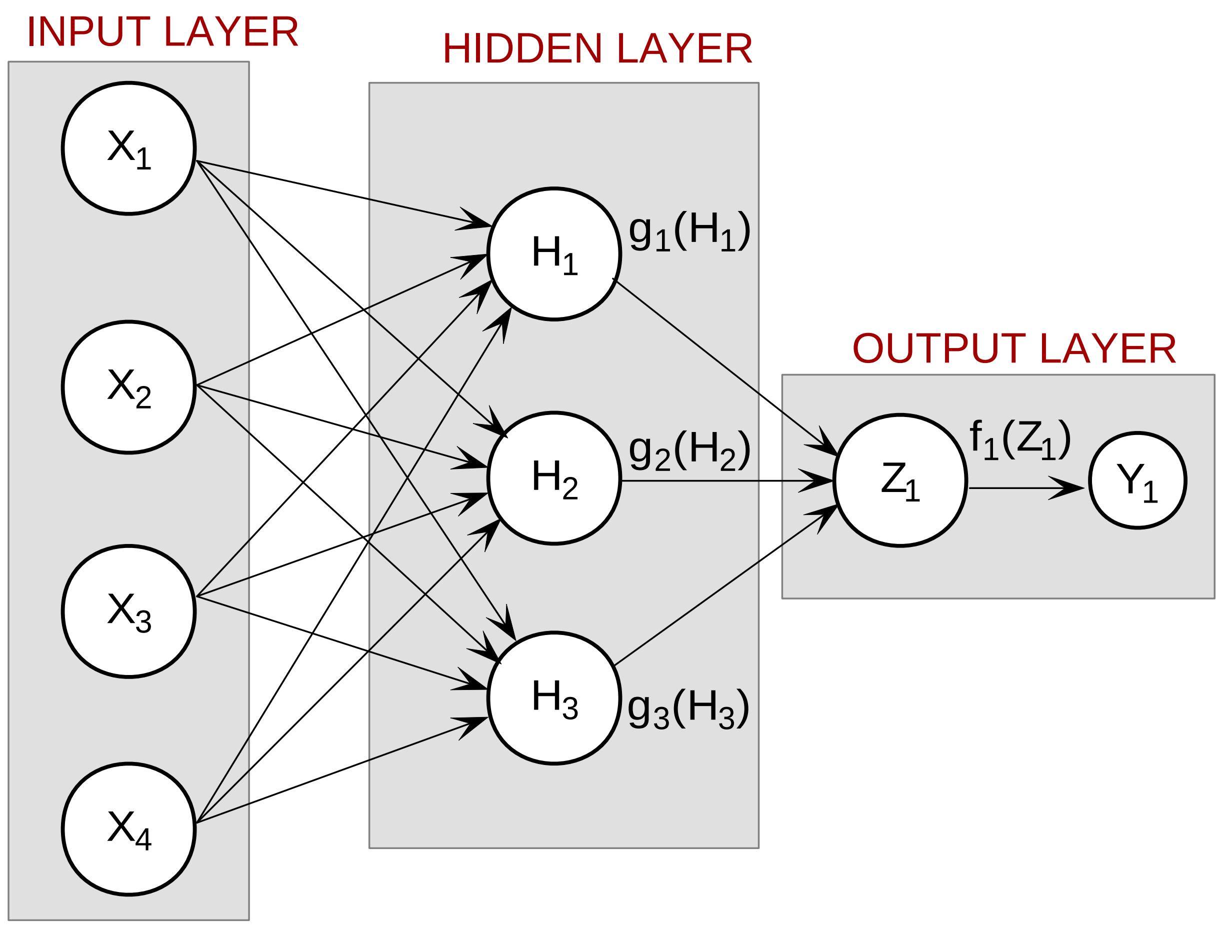
This example trains a 2-layer network using 100 training patterns from one nominal and one continuous input attribute. The nominal attribute has three classifications which are encoded using binary encoding. This results in three binary network input columns. The continuous input attribute is scaled to fall in the interval [0,1].

The network training targets were generated using the relationship:

$y = 10{*}X_1 + 20{*}X_2 + 30{*}X_3 + 2.0{*}X_4$, where

X_1-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.

```
using System;
using Imsl.DataMining.Neural;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

//***************************************************************************
// 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}, {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, 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}, {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, 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, 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}};
   //
   // contAtt[]:  A matrix of values for the continuous training attribute
   //
   private static double[] contAtt = new double[]{4.007054658, 7.10028447,
      4.740350984, 5.714553211, 6.205437459, 2.598930065, 8.65089967,
      5.705787357, 2.513348184, 2.723795955, 4.1829356, 1.93280416,
      0.332941608, 6.745567628, 5.593588463, 7.273544478, 3.162117939,
      4.205381208, 0.16414745, 2.883418275, 0.629342241, 1.082223406,
      8.180324708, 8.004894314, 7.856215418, 7.797143157, 8.350033996,
      3.778254431, 6.964837082, 6.13938006, 0.48610387, 5.686627923,
      8.146173848, 5.879852653, 4.587492779, 0.714028533, 7.56324211,
      8.406012623, 4.225261454, 6.369220241, 4.432772218, 9.52166984,
      7.935791508, 4.557155333, 7.976015058, 4.913538616, 1.473658514,
      2.592338905, 1.386872932, 7.046051685, 1.432128376, 1.153580985,
      5.6561491, 3.31163251, 4.648324851, 5.042514515, 0.657054195,
      7.958308093, 7.557870384, 7.901990083, 5.2363088, 6.95582150,
      8.362167045, 4.875903563, 1.729229471, 4.380370223, 8.527875685,
      2.489198107, 3.711472959, 4.17692681, 5.844828801, 4.825754155,
      5.642267843, 5.339937786, 4.440813223, 1.615143829, 7.542969339,
      8.100542684, 0.98625265, 4.744819569, 8.926039258, 8.813441887,
      7.749383991, 6.551841576, 8.637046998, 4.560281415, 1.386055087,
      0.778869034, 3.883379045, 2.364501589, 9.648737525, 1.21754765,
      3.908879368, 4.253313879, 9.31189696, 3.811953836, 5.78471629,
      3.414486452, 9.345413015, 1.024053777};
   //
   // 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
//
private static double[] outs = new double[]{18.01410932, 24.20056894,
   19.48070197, 21.42910642, 22.41087492, 15.19786013, 27.30179934,
   21.41157471, 15.02669637, 15.44759191, 18.3658712, 13.86560832,
   10.66588322, 23.49113526, 21.18717693, 24.54708896, 16.32423588,
   18.41076242, 10.3282949, 15.76683655, 11.25868448, 12.16444681,
   26.36064942, 26.00978863, 25.71243084, 25.59428631, 26.70006799,
   17.55650886, 23.92967416, 22.27876012, 10.97220774, 21.37325585,
   26.2923477, 21.75970531, 19.17498556, 21.42805707, 35.12648422,
   36.81202525, 28.45052291, 32.73844048, 28.86554444, 39.04333968,
   35.87158302, 29.11431067, 35.95203012, 29.82707723, 22.94731703,
   25.18467781, 22.77374586, 34.09210337, 22.86425675, 22.30716197,
   31.3122982, 26.62326502, 29.2966497, 30.08502903, 21.31410839,
   35.91661619, 35.11574077, 35.80398017, 30.4726176, 33.91164302,
   36.72433409, 29.75180713, 23.45845894, 38.76074045, 47.05575137,
   34.97839621, 37.42294592, 38.35385362, 41.6896576, 39.65150831,
   41.28453569, 40.67987557, 38.88162645, 33.23028766, 45.08593868,
   46.20108537, 31.9725053, 39.48963914, 47.85207852, 47.62688377,
   45.49876798, 43.10368315, 47.274094, 39.1205628, 32.77211017,
   31.55773807, 37.76675809, 34.72900318, 49.29747505, 32.4350953,
   37.81775874, 38.50662776, 48.62379392, 37.62390767, 41.56943258,
   36.8289729, 48.69082603, 32.04810755};
// ***********************************************************************
// MAIN
// ***********************************************************************
[STAThread]
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 indicies
   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
// ************************************************************************
System.Console.Out.WriteLine(
   "--> Training 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:                       " +
      (float) stats[1]);
   System.Console.Out.WriteLine("--> Laplacian Error:           " +
      (float) stats[2]);
   System.Console.Out.WriteLine("--> Scaled Laplacian Error:    " +
      (float) stats[3]);
   System.Console.Out.WriteLine("--> Largest Absolute Residual: " +
      (float) stats[4]);
   System.Console.Out.WriteLine(
      "***********************************************");
   System.Console.Out.WriteLine("");
   // ***********************************************************************
   // OBTAIN AND DISPLAY NETWORK WEIGHTS AND GRADIENTS
   // ***********************************************************************
   System.Console.Out.WriteLine("--> Getting Network Weights and Gradients");
   // 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("--> Network Weights and Gradients:");
   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(
      "***********************************************");
   // ***********************************************************************
```

```
      // SAVE THE TRAINED NETWORK BY SAVING THE SERIALIZED NETWORK OBJECT
      // ******************************************************************
      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[2]=-1.491785 g[2]=-2.611079E-08
```

```
w[3]=1.616918 g[3]=6.182035E-08
w[4]=1.616918 g[4]=6.182035E-08
w[5]=1.616918 g[5]=6.182035E-08
w[6]=4.725622 g[6]=-5.273856E-08
w[7]=4.725622 g[7]=-5.273856E-08
w[8]=4.725622 g[8]=-5.273856E-08
w[9]=6.217407 g[9]=-8.733E-10
w[10]=6.217407 g[10]=-8.733E-10
w[11]=6.217407 g[11]=-8.733E-10
w[12]=1.072258 g[12]=-1.690978E-07
w[13]=1.072258 g[13]=-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
***********************************************

--> Saving Trained Network into FeedForwardNetworkEx1.ser
--> Saving xData into FeedForwardxDataEx1.ser
--> Saving yData into FeedForwardyDataEx1.ser
--> Saving Network Trainer into FeedForwardTrainerEx1.ser
```

Layer Class

Summary

The base class for `Layer`s in a neural network.

`public class Imsl.DataMining.Neural.Layer`

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`.

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. 1026) with this `Layer`.

Parameter

`node` – A `Node` to associate with this `Layer`.

See Also

Imsl.DataMining.Neural.InputLayer (p. 1019), Imsl.DataMining.Neural.HiddenLayer (p. 1020)

InputLayer Class

Summary

Input layer in a neural network.

```
public class Imsl.DataMining.Neural.InputLayer : Layer
```

Property

Nodes

```
override public Imsl.DataMining.Neural.Node[] Nodes {get; }
```

Description

The Perceptrons (p. 1026) in the InputLayer.

Methods

CreateInput

```
virtual public Imsl.DataMining.Neural.InputNode CreateInput()
```

Description

Creates an InputNode in the InputLayer of the neural network.

CreateInputs

```
virtual public Imsl.DataMining.Neural.InputNode[] CreateInputs(int n)
```

Description

Creates a number of InputNodes in this Imsl.DataMining.Neural.Layer (p. 1018) of the neural network.

Parameter

n – An int which specifies the number of InputNodes to be created in this Layer.

Returns

An InputNodearray containing the created InputNodes.

Description

An InputLayer is automatically created by Network.

See Also

Imsl.DataMining.Neural.Network (p. 1148)

HiddenLayer Class

Summary

Hidden layer in a neural network. This is created by a factory method in Imsl.DataMining.Neural.Network (p. 1148).

```
public class Imsl.DataMining.Neural.HiddenLayer :  Layer
```

Methods

CreatePerceptron

```
virtual public Imsl.DataMining.Neural.Perceptron CreatePerceptron()
```

Description

Creates a `Perceptron` in this Imsl.DataMining.Neural.Layer (p. 1018) of the neural network.

The created `Perceptron` uses the logistic activation function and has an initial Bias (p. 1026) value of zero.

CreatePerceptron

```
virtual public Imsl.DataMining.Neural.Perceptron
  CreatePerceptron(Imsl.DataMining.Neural.IActivation activation, double
  bias)
```

Description

Creates a `Perceptron` in this Imsl.DataMining.Neural.Layer (p. 1018) with a specified activation function and bias (p. 1026).

Parameters

`activation` – The `IActivation` object which specifies the activation function to be used.

`bias` – A `double` which specifies the initial value for the `Bias`.

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 Imsl.DataMining.Neural.Layer (p. 1018) with the specified Bias (p. 1026).

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`.

CreatePerceptrons

```
virtual public Imsl.DataMining.Neural.Perceptron[] CreatePerceptrons(int n)
```

Description

Creates a number of `Perceptrons` in this Imsl.DataMining.Neural.Layer (p. 1018) of the neural network.

The created `Perceptrons` use the logistic activation function and have an initial Bias (p. 1026) value of zero.

Parameter

`n` – An `int` which specifies the number of `Perceptrons` to be created.

Returns

An array containing the created `Perceptrons`.

See Also

Imsl.DataMining.Neural.Network.CreateHiddenLayer (p. 1150)

OutputLayer Class

Summary

Output layer in a neural network.

```
public class Imsl.DataMining.Neural.OutputLayer :  Layer
```

Property

Nodes

```
override public Imsl.DataMining.Neural.Node[] Nodes {get; }
```

Description

The Imsl.DataMining.Neural.Perceptron (p. 1026)s in the Imsl.DataMining.Neural.OutputLayer (p. 1022).

This method overrides the method in Imsl.DataMining.Neural.Layer (p. 1018) to return the `Perceptrons` in an `OutputPerceptron` array.

Methods

CreatePerceptron

```
virtual public Imsl.DataMining.Neural.Perceptron CreatePerceptron()
```

Description

Creates a `Perceptron` in this Imsl.DataMining.Neural.Layer (p. 1018) of the neural network. By default, the created `Perceptron` uses the linear activation function and has an initial Bias (p. 1026) value of zero.

CreatePerceptron

```
virtual public Imsl.DataMining.Neural.Perceptron
  CreatePerceptron(Imsl.DataMining.Neural.IActivation activation, double
  bias)
```

Description

Creates a `Perceptron` in this Imsl.DataMining.Neural.Layer (p. 1018) with a specified `Activation` and Bias (p. 1026).

Parameters

`activation` – The `Activation` object which specifies the action function to be used.

`bias` – A `double` which specifies the initial value for the `Bias` for this `Perceptron`.

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 Imsl.DataMining.Neural.Layer (p. 1018) with specified `Activation` and Bias (p. 1026).

Parameters

`n` – An `int` which specifies the number of `Perceptrons` to be created.

`activation` – The `Activation` 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`.

CreatePerceptrons

```
virtual public Imsl.DataMining.Neural.Perceptron[] CreatePerceptrons(int n)
```

Description

Creates a number of `Perceptrons` in this Imsl.DataMining.Neural.Layer (p. 1018) of the neural network. By default, they will use linear activation and a zero initial Bias (p. 1026).

Parameter

`n` – An `int` which specifies the number of `Perceptrons` to be created in this `Layer`.

Returns

An array containing the created `Perceptrons`.

Description

An empty `OutputLayer` is automatically created by
Imsl.DataMining.Neural.FeedForwardNetwork (p. 1004).

See Also

Imsl.DataMining.Neural.Network (p. 1148)

Node Class

Summary

A `Node` in a neural network.

`public class Imsl.DataMining.Neural.Node`

Property

Layer

`virtual public Imsl.DataMining.Neural.Layer Layer {get; }`

Description

The `Layer` in which this `Node` exists.

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`.

Description

`Node` is an abstract class that serves as the base class for the concrete classes `InputNode` and `Perceptron`.

See Also

Imsl.DataMining.Neural.InputNode (p. 1025), Imsl.DataMining.Neural.Perceptron (p. 1026)

InputNode Class

Summary

A `Node` in the Imsl.DataMining.Neural.InputLayer (p. 1019).

`public class Imsl.DataMining.Neural.InputNode : Node`

Methods

GetValue

`override public double GetValue()`

Description

Returns the value of this Imsl.DataMining.Neural.Node (p. 1024).

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. 1024).

Parameter

`node` – A `double` which specifies the new value of this `InputNode`.

Description

InputNodes are not created directly. Instead factory methods in InputLayer are used to create InputNodes within the InputLayer. For example,
Imsl.DataMining.Neural.InputLayer.CreateInput (p. 1020) creates a single InputNode.

See Also

Feed Forward Class Example 1

Perceptron Class

Summary

A Perceptron node in a neural network.

```
public class Imsl.DataMining.Neural.Perceptron :  Node
```

Properties

Activation

```
virtual public Imsl.DataMining.Neural.IActivation Activation {get; set; }
```

Description

The activation function.

Bias

```
virtual public double Bias {get; set; }
```

Description

The Bias for this perceptron.

The Bias has a default value of 0.

Description

Perceptrons are created by factory methods in a Layer (p. 1018). Each Perceptron has an Activation (p. 1026) function (g) and a bias (μ) (p. 1026). The value of a Perceptron is given by $g(\sum_i w_i X_i + \mu)$, where X_is are the values of nodes input to this Perceptron with Weight (w_i) (p. 1029).

Network (p. 1148) 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.

OutputPerceptron Class

Summary

A `Perceptron` in the output layer.

```
public class Imsl.DataMining.Neural.OutputPerceptron :  Perceptron
```

Method

GetValue

```
override public double GetValue()
```

Description

Returns the value of the output perceptron determined using the current Imsl.DataMining.Neural.Network (p. 1148) state and inputs.

Returns

A `double` value of the output perceptron determined using the current `Network` state and inputs.

Description

`OutputPerceptron`s are created by factory methods in `Outputlayer`.

`OutputPerceptron`s are not created directly. Instead factory methods in `OutputLayer` are used to create `OutputPerceptron`s within the `OutputLayer`. For example, `OutputLayer.createPerceptron()` creates a single `OutputPerceptron`.

See Also

Imsl.DataMining.Neural.OutputLayer (p. 1022)

IActivation Interface

Summary

Interface implemented by perceptron activation functions.

```
public interface Imsl.DataMining.Neural.IActivation
```

Methods

Derivative

`abstract public double Derivative(double x, double y)`

Description

Returns the value of the derivative of the activation function.

y is not mathematically required, but can sometimes be used to more quickly compute the derivative.

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.

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.

Description

Standard activation functions are defined as static members of this interface. New activation functions can be defined by implementing a method, `g(double x)`, returning the value and a method, `derivative(double x, double y)`, returning the derivative of `g` evaluated at `x` where $y = g(x)$.

See Also

Imsl.DataMining.Neural.Perceptron (p. 1026)

Link Class

Summary

A link in a neural network.

```
public class Imsl.DataMining.Neural.Link
```

Properties

From

`virtual public Imsl.DataMining.Neural.Node From {get; }`

Description

The origination `Node` for this `Link`.

To

`virtual public Imsl.DataMining.Neural.Node To {get; }`

Description

The destination `Node` for this `Link`.

Weight

`virtual public double Weight {get; set; }`

Description

The *weight* for this `Link`.

Description

`Link` objects are not created directly. Instead, they are created by factory methods in `FeedForwardNetwork`.

The most useful method is LinkAll() (p. 1008) which creates `Link` objects connecting every Imsl.DataMining.Neural.Node (p. 1024) in each Imsl.DataMining.Neural.Layer (p. 1018) to every `Node` in the next `Layer`.

The method Link(node,node) (p. 1007) creates a `Link` from a `Node` to any `Node` in a later `Layer`.

The method FindLink(Node,Node) (p. 1006) returns the `Link` connecting two `Nodes` in the Imsl.DataMining.Neural.Network (p. 1148).

The method Remove(Link) (p. 1008) removes a `Link` from the `Network`.

Each `Link` object contains an Weight (p. 1029). `Weights` are used in computing Perceptron (p. 1026) values.

See Also

Imsl.DataMining.Neural.FeedForwardNetwork (p. 1004)

ITrainer Interface

Summary

Interface implemented by classes used to train an Imsl.DataMining.Neural.Network (p. 1148).

```
public interface Imsl.DataMining.Neural.ITrainer
```

Properties

ErrorGradient

```
abstract public double[] ErrorGradient {get; }
```

Description

The value of the gradient of the error function with respect to the Weights (p. 1029).

Before training, `null` is returned.

ErrorStatus

```
abstract public int ErrorStatus {get; }
```

Description

The error status.

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.

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.

The number of columns in *xData* must equal the number of nodes in the input layer. Each row of *xData* contains a training pattern.

The number of columns in *yData* must equal the number of perceptrons in the output layer. Each row of *yData* contains a training pattern.

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.

Description

The method `Train` is used to adjust the Weights (p. 1029) 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.

QuasiNewtonTrainer Class

Summary

Trains an Imsl.DataMining.Neural.Network (p. 1148) using the quasi-Newton method, `MinUnconMultiVar`.

```
public class Imsl.DataMining.Neural.QuasiNewtonTrainer :
Imsl.DataMining.Neural.ITrainer, ICloneable
```

Field

```
SUM_OF_SQUARES
public Imsl.DataMining.Neural.QuasiNewtonTrainer.IError SUM_OF_SQUARES
```

Description

Compute the sum of squares error.

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.

Error

```
virtual public Imsl.DataMining.Neural.QuasiNewtonTrainer.IError Error {get;
  set; }
```

Description

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. 1029).

Before training, `null` is returned.

ErrorStatus

```
virtual public int ErrorStatus {get; }
```

Description

The error status from the trainer.

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.

Error Status	Condition
0	No error occurred during training.
1	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.
2	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.
3	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.
4	Optimizer threw a `MinUnconMultiVar.FalseConvergenceException`.
5	Optimizer threw a `MinUnconMultiVar.MaxIterationsException`.
6	Optimizer threw a `MinUnconMultiVar.UnboundedBelowException`.

See Also: Imsl.Math.FalseConvergenceException (p. 1174), Imsl.Math.MaxIterationsException (p. 1180), Imsl.Math.UnboundedBelowException (p. 1199)

ErrorValue

`virtual public double ErrorValue {get; }`

Description

The final value of the error function.

Before training, `NaN` is returned.

FalseConvergenceTolerance

`virtual public double FalseConvergenceTolerance {get; set; }`

Description

The false convergence tolerance for the Imsl.DataMining.Neural.ITrainer (p. 1030).

Default: 2.220446049250313080e-14.

See Also: Imsl.Math.MinUnconMultiVar.FalseConvergenceTolerance (p. 128)

GradientTolerance

`virtual public double GradientTolerance {get; set; }`

Description

The gradient tolerance.

Default: cube root of machine precision.

See Also: Imsl.Math.MinUnconMultiVar.GradientTolerance (p. 128)

MaximumStepsize

```
virtual public double MaximumStepsize {get; set; }
```

Description

The maximum step size.

The value of `MaximumStepsize` will be equal to -999.0 if the default value is to be used and the Train (p. 1036) method has not been called.

See Also: (p. 129)

MaximumTrainingIterations

```
virtual public int MaximumTrainingIterations {get; set; }
```

Description

The maximum number of iterations to use in a training.

Default: 100.

See Also: (p. 129)

ParallelMode

```
virtual protected internal System.Collections.ArrayList[] ParallelMode {set;
  }
```

Description

The trainer to be used in multi-threaded EpochTainer.

RelativeTolerance

```
virtual public double RelativeTolerance {get; set; }
```

Description

The relative tolerance.

It must be in the interval [0,1]. Its default value is 3.66685e-11.

See Also: Imsl.Math.MinUnconMultiVar.RelativeTolerance (p. 129)

StepTolerance

```
virtual public double StepTolerance {get; set; }
```

Description

The scaled step tolerance.

The second stopping criterion for Imsl.Math.MinUnconMultiVar (p. 127), the optimizer used by this Imsl.DataMining.Neural.ITrainer (p. 1030), is that the scaled distance between the last two steps be less than the step tolerance.

Default: 3.66685e-11.

See Also: Imsl.Math.MinUnconMultiVar.StepTolerance (p. 129)

TrainingIterations

```
virtual public int TrainingIterations {get; }
```

Description

The number of iterations used during training.

See Also: (p. 128)

UseBackPropagation

```
virtual public bool UseBackPropagation {get; set; }
```

Description

Specify the use of the back propagation algorithm for gradient calculations during network training.

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.

The default is to compute the sum of squares error, `SUM_OF_SQUARES`.

Parameter

`error` – The `IError` object containing the function to be used to compute the network error.

Train

```
virtual public void Train(Imsl.DataMining.Neural.Network network, double[,]
  xData, double[,] yData)
```

Description

Trains the neural network using supplied training patterns.

The number of columns in *xData* must equal the number of Nodes (p. 1024) in the input layer.

The number of columns in *yData* must equal the number of Perceptrons (p. 1026) in the output layer.

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 (p. 1029) in the `Network`.

Parameters

`network` – The `Network` to be trained.

`xData` – An input `double` matrix containing training patterns.

`yData` – An output `double` matrix containing output training patterns.

See Also

Imsl.Math.MinUnconMultiVar (p. 127)

QuasiNewtonTrainer.IError Interface

Summary

Error function to be minimized by trainer.

```
public interface Imsl.DataMining.Neural.QuasiNewtonTrainer.IError
```

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.

Description

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, $de/d\hat{y}$.

LeastSquaresTrainer Class

Summary

Trains a Imsl.DataMining.Neural.FeedForwardNetwork (p. 1004) using a Levenberg-Marquardt algorithm for minimizing a sum of squares error.

```
public class Imsl.DataMining.Neural.LeastSquaresTrainer :
Imsl.DataMining.Neural.ITrainer
```

Properties

EpochNumber

`virtual protected internal int EpochNumber {set; }`

Description

The epoch number for the trainer.

ErrorGradient

`virtual public double[] ErrorGradient {get; }`

Description

The value of the *gradient* of the error function with respect to the Weights (p. 1029).

Before training, `null` is returned.

ErrorStatus

`virtual public int ErrorStatus {get; }`

Description

The error status from the trainer.

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.

Value	Meaning
0	All convergence tests were met.
1	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.
2	Scaled actual and predicted reductions in the function are less than or equal to the relative function convergence tolerance `RelativeTolerance`.
3	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.
4	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.
5	Too many iterations required.

ErrorValue

`virtual public double ErrorValue {get; }`

Description

The final value of the error function.

Before training, `NaN` is returned.

FalseConvergenceTolerance

`virtual public double FalseConvergenceTolerance {get; set; }`

Description

The false convergence tolerance.

Default: 1.0e-14.

See Also: NonlinLeastSquares.FalseConvergenceTolerance (p. 135)

GradientTolerance

`virtual public double GradientTolerance {get; set; }`

Description

The *gradient* tolerance.

Default: 2.0e-5.

See Also: NonlinLeastSquares.GradientTolerance (p. 136)

InitialTrustRegion

`virtual public double InitialTrustRegion {get; set; }`

Description

The initial trust region.

Default: 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. 136)

MaximumStepsize

`virtual public double MaximumStepsize {get; set; }`

Description

The maximum step size.

Default: $10^3||w||_2$, where w are the values of the Weights (p. 1029) 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. 136)

MaximumTrainingIterations

`virtual public int MaximumTrainingIterations {get; set; }`

Description

The maximum number of iterations used by the nonlinear least squares solver.

Its default value is 1000.

See Also: NonlinLeastSquares.RelativeTolerance (p. 136)

ParallelMode

`virtual protected internal System.Collections.ArrayList[] ParallelMode {set; }`

Description

The trainer to be used in multi-threaded EpochTainer.

RelativeTolerance

`virtual public double RelativeTolerance {get; set; }`

Description

The relative tolerance.

It must be in the interval [0,1]. Its default value is 1.0e-20.

See Also: NonlinLeastSquares.RelativeTolerance (p. 136)

StepTolerance

`virtual public double StepTolerance {get; set; }`

Description

The step tolerance used to step between Weights (p. 1029).

Default: 1.0e-5.

See Also: NonlinLeastSquares.StepTolerance (p. 136)

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.

Each row of *xData* and *yData* contains a training pattern. These number of rows in two arrays must be equal.

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 (p. 1024) in the Imsl.DataMining.Neural.InputLayer (p. 1019).

`yData` – A `double` matrix which contains the output training patterns. The number of columns in *yData* must equal the number of Perceptrons (p. 1026) in the Imsl.DataMining.Neural.OutputLayer (p. 1022).

See Also

NonlinLeastSquares (p. 134)

EpochTrainer Class

Summary

Two-stage training using randomly selected training patterns in stage I.

```
public class Imsl.DataMining.Neural.EpochTrainer :
Imsl.DataMining.Neural.ITrainer
```

Properties

EpochSize

`virtual public int EpochSize {get; set; }`

Description

The number of randomly selected training patterns in each stage I epoch.

ErrorGradient

`virtual public double[] ErrorGradient {get; }`

Description

The value of the *gradient* of the error function with respect to the weights (p. 1149).

Before training, `null` is returned.

ErrorStatus

`virtual public int ErrorStatus {get; }`

Description

The training error status.

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.

NumberOfEpochs

`virtual public int NumberOfEpochs {get; set; }`

Description

The number of epochs used during stage I training.

Random

```
virtual public Imsl.Stat.Random Random {get; set; }
```

Description

The random number generator used to perturb the stage I guesses.

Stage1Trainer

```
virtual protected internal Imsl.DataMining.Neural.ITrainer Stage1Trainer
  {get; }
```

Description

The stage 1 trainer.

Stage2Trainer

```
virtual protected internal Imsl.DataMining.Neural.ITrainer Stage2Trainer
  {get; }
```

Description

The stage 1 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

`virtual public void SetRandomSamples(Imsl.Stat.Random randomA, Imsl.Stat.Random randomB)`

Description

Sets the random number generators used to select random training patterns in stage I.

The two random number generators should be independent.

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*.

Train

`virtual public void Train(Imsl.DataMining.Neural.Network network, double[,] xData, double[,] yData)`

Description

Trains the neural network using supplied training patterns.

Each row of *xData* and *yData* contains a training pattern. These number of rows in two arrays must be equal.

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 (p. 1024) in the Imsl.DataMining.Neural.InputLayer (p. 1019).

`yData` – A `double` containing the output training patterns. The number of columns in *yData* must equal the number of Perceptrons (p. 1026) in the Imsl.DataMining.Neural.OutputLayer (p. 1022).

Description

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. 1149) 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.

BinaryClassification Class

Summary

Classifies patterns into two classes.

`public class Imsl.DataMining.Neural.BinaryClassification`

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.

Network

`virtual public Imsl.DataMining.Neural.Network Network {get; }`

Description

The network being used for classification.

Constructor

BinaryClassification

`public BinaryClassification(Imsl.DataMining.Neural.Network network)`

Description

Creates a binary classifier.

Parameter

`network` – Is the neural network used for classification. Its output perceptron should use the logistic activation function.

Methods

ComputeStatistics

`virtual public double[] ComputeStatistics(double[,] xData, int[] yData)`

Description

Computes the classification error statistics for the supplied network patterns and their associated classifications.

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.

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 number of columns in `yData` must equal the number of `Perceptrons` in the `OutputLayer`.

Returns

A two-element `double` array containing the binary cross-entropy error and the classification error rate.

PredictedClass

`virtual public int PredictedClass(double[] x)`

Description

Calculates the classification probablities for the input pattern `x`, and returns either 0 or 1 identifying the class with the highest probability.

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.

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.

Probabilities

`virtual public double[] Probabilities(double[] x)`

Description

Returns classification probabilities for the input pattern `x`.

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 output perceptron'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.

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.

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 Imsl.DataMining.Neural.BinaryClassification.Error (p. 1045) 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.

Description

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 $P(C_2) = 1 - P(C_1)$.

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 System.Runtime.Serialization.Formatters.Binary;
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, 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 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2};
```

```
/* 3 classifications */
private static int[] x3 = new int[]{1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 1,
   1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 1,
   1, 1, 1, 2, 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};

// **********************************************************************
// MAIN
// **********************************************************************
[STAThread]
public static void  Main(System.String[] args)
{
   double[,] xData; // Input  Attributes for Trainer
   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.FalseConvergenceTolerance = 1.0e-20;
trainer.StepTolerance = 1.0e-20;
trainer.RelativeTolerance = 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
System.Console.Out.WriteLine(
   "**********************************************");
System.Console.Out.WriteLine("--> Cross-Entropy Error:      " +
   (float)statsClass[0]);
System.Console.Out.WriteLine("--> Classification Error:     " +
   (float)statsClass[1]);
System.Console.Out.WriteLine(
   "**********************************************");
System.Console.Out.WriteLine("");


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
**********************************************

          Weights              Gradients
 0   1.2162782442665     -1.82357413516679E-12
 1  -7.10104582036137     4.00527512119464E-13
 2  -4.48633964224305     9.39571675391483E-13
 3  -2.60725959847226     4.68033472575564E-09
 4   4.29051046747984    -1.02798278799989E-09
 5   4.48156618131766    -2.41147856556375E-09
 6   2.08243325148918     4.67851115006368E-09
 7  -6.8692099513798     -1.02758226014584E-09
 8  -4.71797133795929    -2.41053899308624E-09
 9  -3.15604455817262     1.55679543763229E-18
10   6.63883077983807    -3.41932577068912E-19
11   4.87196888567521    -8.02117593888885E-19
12  -2.30032598691343    -1.82357569033459E-12
13  -1.68298220558961     4.0052785369455E-13
14   1.57459851986179     9.39572476670461E-13
15  -0.26445116158594     5.9515904744093E-10
16  -0.649412990131875   -1.30719978963237E-10
17  -0.124557044207741   -3.06647573325737E-10
18   0.125744106228649    4.08517567986988E-09
19  -0.550795793591825   -8.97262809378226E-10
20   0.213785518241935   -2.10483099303929E-09
21  -0.256460169761589    1.46860577630575E-15
22   0.719269734080646   -3.22562711613705E-16
23   0.181431096607655   -7.56679074967797E-16
24   0.708887793360319    4.98153020829334E-10
25  -0.137808725484545   -1.09413698209379E-10
26  -0.0714270451579146  -2.56666542563753E-10
27  -1.69080392381497    -1.82357569086926E-12
28  -1.05894442754156     4.00527853811983E-13
29   0.916792419031426    9.3957247694594E-13
30   0.848401861786795    4.18218023787623E-09
31   0.809968676184518   -9.1856876756949E-10
32  -0.621014362841794   -2.15481126712248E-09
33   3.92683360378712    -1.53010895435583E-09
34   3.92683360363108    -1.53010895365419E-09
35  -0.862484756249916    6.2509576909593E-10
36  -0.862484756609361    6.25095768809291E-10
37  -2.02324740009297    -1.05620718601144E-09
38  -2.02324740027673    -1.05620718552711E-09
39   1.26293424026308    -5.17748567971874E-09
```

40	1.26293423968395	-5.17748567952834E-09
41	-2.95381709009517	4.67851115162047E-09
42	3.47670654419506	-1.02758226048777E-09
43	-1.14723052699407	-2.41053899388836E-09
44	-3.57249695789483	5.95710389698464E-10
45	-3.57249695784057	5.95710389425299E-10
46	5.5108803426582	4.71687575415149E-10

				Forecast		
	X1	X2	X3	X4	Expected	Predicted
0	1	1	1	1	1	1
1	1	1	1	2	1	1
2	1	1	1	3	1	1
3	1	1	1	4	1	1
4	1	1	2	1	1	1
5	1	1	2	2	1	1
6	1	1	2	3	1	1
7	1	1	2	4	1	1
8	1	1	3	1	1	1
9	1	1	3	2	1	1
10	1	1	3	3	1	1
11	1	1	3	4	1	1
12	1	2	1	1	0	0
13	1	2	1	2	0	0
14	1	2	1	3	0	0
15	1	2	1	4	0	0
16	1	2	2	1	0	0
17	1	2	2	2	0	0
18	1	2	2	3	0	0
19	1	2	2	4	0	0
20	1	2	3	1	0	0
21	1	2	3	2	0	0
22	1	2	3	3	0	0
23	1	2	3	4	0	0
24	2	1	1	1	0	0
25	2	1	1	2	0	0
26	2	1	1	3	0	0
27	2	1	1	4	0	0
28	2	1	2	1	0	0
29	2	1	2	2	0	0
30	2	1	2	3	0	0
31	2	1	2	4	0	0
32	2	1	3	1	0	0
33	2	1	3	2	0	0
34	2	1	3	3	0	0
35	2	1	3	4	0	0
36	2	2	1	1	0	0
37	2	2	1	2	0	0
38	2	2	1	3	0	0
39	2	2	1	4	0	0
40	2	2	2	1	0	0
41	2	2	2	2	0	0
42	2	2	2	3	0	0
43	2	2	2	4	0	0
44	2	2	3	1	0	0
45	2	2	3	2	0	0

```
46  2   2   3   3      0          0
47  2   2   3   4      0          0

*********************************************
--> Cross-Entropy Error:      4.720552E-10
--> Classification Error:     0
*********************************************

***************Time:  0.047
trainer.getErrorValue = 4.72055172799E-10
```

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.

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 atribute 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.

```
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=O, 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, 1},
      new int[]{0, 0, 0, 0, 1, 1, 1, 0, 1}, new int[]{0, 0, 0, 0, 1, 1, 1, 1, 0},
      new int[]{0, 0, 0, 0, 1, 1, 1, 2, 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, 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, 2, 1, 1, 1, 2}, new int[]{0, 0, 0, 0, 2, 1, 1, 2, 1},
new int[]{0, 0, 0, 0, 2, 1, 2, 1, 1}, new int[]{0, 0, 0, 1, 0, 1, 0, 1, 1},
new int[]{0, 0, 0, 1, 0, 1, 1, 0, 1}, new int[]{0, 0, 0, 1, 0, 1, 1, 1, 0},
new int[]{0, 0, 0, 1, 0, 1, 1, 2, 2}, new int[]{0, 0, 0, 1, 0, 1, 2, 1, 2},
new int[]{0, 0, 0, 1, 0, 1, 2, 2, 1}, new int[]{0, 0, 0, 1, 0, 2, 1, 1, 2},
new int[]{0, 0, 0, 1, 0, 2, 1, 2, 1}, new int[]{0, 0, 0, 1, 0, 2, 2, 1, 1},
new int[]{0, 0, 0, 1, 1, 0, 0, 1, 1}, new int[]{0, 0, 0, 1, 1, 0, 1, 0, 1},
new int[]{0, 0, 0, 1, 1, 0, 1, 1, 0}, new int[]{0, 0, 0, 1, 1, 0, 1, 2, 2},
new int[]{0, 0, 0, 1, 1, 0, 2, 1, 2}, new int[]{0, 0, 0, 1, 1, 0, 2, 2, 1},
new int[]{0, 0, 0, 1, 1, 2, 0, 1, 2}, new int[]{0, 0, 0, 1, 1, 2, 0, 2, 1},
new int[]{0, 0, 0, 1, 1, 2, 1, 0, 2}, new int[]{0, 0, 0, 1, 1, 2, 1, 2, 0},
new int[]{0, 0, 0, 1, 1, 2, 2, 0, 1}, new int[]{0, 0, 0, 1, 1, 2, 2, 1, 0},
new int[]{0, 0, 0, 1, 1, 2, 2, 2, 2}, new int[]{0, 0, 0, 1, 2, 0, 1, 1, 2},
new int[]{0, 0, 0, 1, 2, 0, 1, 2, 1}, new int[]{0, 0, 0, 1, 2, 0, 2, 1, 1},
new int[]{0, 0, 0, 1, 2, 1, 0, 1, 2}, new int[]{0, 0, 0, 1, 2, 1, 0, 2, 1},
new int[]{0, 0, 0, 1, 2, 1, 1, 0, 2}, new int[]{0, 0, 0, 1, 2, 1, 1, 2, 0},
new int[]{0, 0, 0, 1, 2, 1, 2, 0, 1}, new int[]{0, 0, 0, 1, 2, 1, 2, 1, 0},
new int[]{0, 0, 0, 1, 2, 1, 2, 2, 2}, new int[]{0, 0, 0, 1, 2, 2, 0, 1, 1},
new int[]{0, 0, 0, 1, 2, 2, 1, 0, 1}, new int[]{0, 0, 0, 1, 2, 2, 1, 1, 0},
new int[]{0, 0, 0, 1, 2, 2, 1, 2, 2}, new int[]{0, 0, 0, 1, 2, 2, 2, 1, 2},
new int[]{0, 0, 0, 1, 2, 2, 2, 2, 1}, new int[]{0, 0, 0, 2, 0, 1, 1, 1, 2},
new int[]{0, 0, 0, 2, 0, 1, 1, 2, 1}, new int[]{0, 0, 0, 2, 0, 1, 2, 1, 1},
new int[]{0, 0, 0, 2, 1, 0, 1, 1, 2}, new int[]{0, 0, 0, 2, 1, 0, 1, 2, 1},
new int[]{0, 0, 0, 2, 1, 0, 2, 1, 1}, new int[]{0, 0, 0, 2, 1, 1, 0, 1, 2},
new int[]{0, 0, 0, 2, 1, 1, 0, 2, 1}, new int[]{0, 0, 0, 2, 1, 1, 1, 0, 2},
new int[]{0, 0, 0, 2, 1, 1, 1, 2, 0}, new int[]{0, 0, 0, 2, 1, 1, 2, 0, 1},
new int[]{0, 0, 0, 2, 1, 1, 2, 1, 0}, new int[]{0, 0, 0, 2, 1, 1, 2, 2, 2},
new int[]{0, 0, 0, 2, 1, 2, 0, 1, 1}, new int[]{0, 0, 0, 2, 1, 2, 1, 0, 1},
new int[]{0, 0, 0, 2, 1, 2, 1, 1, 0}, new int[]{0, 0, 0, 2, 1, 2, 1, 2, 2},
new int[]{0, 0, 0, 2, 1, 2, 2, 1, 2}, new int[]{0, 0, 0, 2, 1, 2, 2, 2, 1},
new int[]{0, 0, 0, 2, 2, 1, 0, 1, 1}, new int[]{0, 0, 0, 2, 2, 1, 1, 0, 1},
new int[]{0, 0, 0, 2, 2, 1, 1, 1, 0}, new int[]{0, 0, 0, 2, 2, 1, 1, 2, 2},
new int[]{0, 0, 0, 2, 2, 1, 2, 1, 2}, new int[]{0, 0, 0, 2, 2, 1, 2, 2, 1},
new int[]{0, 0, 0, 2, 2, 2, 1, 1, 2}, new int[]{0, 0, 0, 2, 2, 2, 1, 2, 1},
new int[]{0, 0, 0, 2, 2, 2, 2, 1, 1}, new int[]{0, 0, 1, 0, 0, 1, 1, 1, 0},
new int[]{0, 0, 1, 0, 1, 0, 0, 1, 1}, new int[]{0, 0, 1, 0, 1, 1, 0, 1, 0},
new int[]{0, 0, 1, 0, 1, 1, 0, 2, 2}, new int[]{0, 0, 1, 0, 1, 2, 0, 1, 2},
new int[]{0, 0, 1, 0, 1, 2, 0, 2, 1}, new int[]{0, 0, 1, 0, 2, 1, 0, 1, 2},
new int[]{0, 0, 1, 0, 2, 2, 0, 1, 1}, new int[]{0, 0, 1, 1, 0, 0, 1, 0, 1},
new int[]{0, 0, 1, 1, 0, 0, 1, 1, 0}, new int[]{0, 0, 1, 1, 0, 1, 0, 1, 0},
new int[]{0, 0, 1, 1, 0, 1, 1, 0, 0}, new int[]{0, 0, 1, 1, 0, 1, 2, 0, 2},
new int[]{0, 0, 1, 1, 0, 1, 2, 2, 0}, new int[]{0, 0, 1, 1, 0, 2, 1, 0, 2},
new int[]{0, 0, 1, 1, 0, 2, 1, 2, 0}, new int[]{0, 0, 1, 1, 0, 2, 2, 0, 1},
new int[]{0, 0, 1, 1, 0, 2, 2, 1, 0}, new int[]{0, 0, 1, 2, 0, 1, 1, 0, 2},
new int[]{0, 0, 1, 2, 0, 1, 1, 2, 0}, new int[]{0, 0, 1, 2, 0, 1, 2, 1, 0},
new int[]{0, 0, 1, 2, 0, 2, 1, 0, 1}, new int[]{0, 0, 1, 2, 0, 2, 1, 1, 0},
new int[]{0, 0, 2, 0, 1, 1, 0, 1, 2}, new int[]{0, 0, 2, 0, 1, 1, 0, 2, 1},
new int[]{0, 0, 2, 0, 1, 2, 0, 1, 1}, new int[]{0, 0, 2, 0, 2, 1, 0, 1, 1},
new int[]{0, 0, 2, 1, 0, 1, 1, 0, 2}, new int[]{0, 0, 2, 1, 0, 1, 1, 2, 0},
new int[]{0, 0, 2, 1, 0, 1, 2, 0, 1}, new int[]{0, 0, 2, 1, 0, 1, 2, 1, 0},
new int[]{0, 0, 2, 1, 0, 2, 1, 0, 1}, new int[]{0, 0, 2, 1, 0, 2, 1, 1, 0},
new int[]{0, 0, 2, 2, 0, 1, 1, 0, 1}, new int[]{0, 0, 2, 2, 0, 1, 1, 1, 0},
new int[]{0, 1, 0, 0, 0, 1, 0, 1, 1}, new int[]{0, 1, 0, 0, 0, 1, 1, 1, 0},
new int[]{0, 1, 0, 0, 1, 1, 0, 0, 1}, new int[]{0, 1, 0, 0, 1, 1, 0, 2, 2},
```

```
new int[]{0, 1, 0, 0, 1, 2, 0, 2, 1}, new int[]{0, 1, 0, 0, 2, 1, 0, 1, 2},
new int[]{0, 1, 0, 0, 2, 1, 0, 2, 1}, new int[]{0, 1, 0, 0, 2, 2, 0, 1, 1},
new int[]{0, 1, 0, 1, 0, 0, 0, 1, 1}, new int[]{0, 1, 0, 1, 0, 0, 1, 1, 0},
new int[]{0, 1, 0, 1, 0, 1, 0, 0, 1}, new int[]{0, 1, 0, 1, 0, 1, 0, 1, 0},
new int[]{0, 1, 0, 1, 0, 1, 0, 2, 2}, new int[]{0, 1, 0, 1, 0, 1, 1, 0, 0},
new int[]{0, 1, 0, 1, 0, 1, 2, 2, 0}, new int[]{0, 1, 0, 1, 0, 2, 0, 1, 2},
new int[]{0, 1, 0, 1, 0, 2, 0, 2, 1}, new int[]{0, 1, 0, 1, 0, 2, 1, 2, 0},
new int[]{0, 1, 0, 1, 0, 2, 2, 1, 0}, new int[]{0, 1, 0, 1, 1, 0, 1, 0, 0},
new int[]{0, 1, 0, 1, 1, 0, 2, 2, 0}, new int[]{0, 1, 0, 1, 2, 0, 1, 2, 0},
new int[]{0, 1, 0, 1, 2, 0, 2, 1, 0}, new int[]{0, 1, 0, 2, 0, 1, 0, 1, 2},
new int[]{0, 1, 0, 2, 0, 1, 0, 2, 1}, new int[]{0, 1, 0, 2, 0, 1, 1, 2, 0},
new int[]{0, 1, 0, 2, 0, 1, 2, 1, 0}, new int[]{0, 1, 0, 2, 0, 2, 0, 1, 1},
new int[]{0, 1, 0, 2, 0, 2, 1, 1, 0}, new int[]{0, 1, 0, 2, 1, 0, 1, 2, 0},
new int[]{0, 1, 0, 2, 2, 0, 1, 1, 0}, new int[]{0, 1, 1, 0, 0, 0, 0, 1, 1},
new int[]{0, 1, 1, 0, 0, 0, 1, 0, 1}, new int[]{0, 1, 1, 0, 0, 0, 1, 1, 0},
new int[]{0, 1, 1, 0, 0, 0, 1, 2, 2}, new int[]{0, 1, 1, 0, 0, 0, 2, 1, 2},
new int[]{0, 1, 1, 0, 0, 0, 2, 2, 1}, new int[]{0, 1, 1, 0, 0, 1, 0, 1, 0},
new int[]{0, 1, 1, 0, 0, 1, 0, 2, 2}, new int[]{0, 1, 1, 0, 0, 1, 1, 0, 0},
new int[]{0, 1, 1, 0, 0, 1, 2, 2, 0}, new int[]{0, 1, 1, 0, 0, 2, 0, 1, 2},
new int[]{0, 1, 1, 0, 0, 2, 0, 2, 1}, new int[]{0, 1, 1, 0, 0, 2, 1, 2, 0},
new int[]{0, 1, 1, 0, 0, 2, 2, 1, 0}, new int[]{0, 1, 1, 0, 1, 0, 0, 0, 1},
new int[]{0, 1, 1, 0, 1, 0, 0, 2, 2}, new int[]{0, 1, 1, 0, 1, 1, 0, 0, 0},
new int[]{0, 1, 1, 0, 1, 2, 0, 0, 2}, new int[]{0, 1, 1, 0, 1, 2, 0, 2, 0},
new int[]{0, 1, 1, 0, 2, 0, 0, 1, 2}, new int[]{0, 1, 1, 0, 2, 0, 0, 2, 1},
new int[]{0, 1, 1, 0, 2, 1, 0, 0, 2}, new int[]{0, 1, 1, 0, 2, 1, 0, 2, 0},
new int[]{0, 1, 1, 0, 2, 2, 0, 0, 1}, new int[]{0, 1, 1, 0, 2, 2, 0, 1, 0},
new int[]{0, 1, 1, 0, 2, 2, 0, 2, 2}, new int[]{0, 1, 1, 1, 0, 0, 0, 1, 0},
new int[]{0, 1, 1, 1, 0, 0, 1, 0, 0}, new int[]{0, 1, 1, 1, 0, 0, 2, 2, 0},
new int[]{0, 1, 1, 1, 0, 1, 0, 0, 0}, new int[]{0, 1, 1, 1, 0, 2, 0, 2, 0},
new int[]{0, 1, 1, 1, 0, 2, 2, 0, 0}, new int[]{0, 1, 1, 1, 1, 0, 0, 0, 0},
new int[]{0, 1, 1, 1, 2, 2, 0, 0, 0}, new int[]{0, 1, 1, 2, 0, 0, 1, 2, 0},
new int[]{0, 1, 1, 2, 0, 0, 2, 1, 0}, new int[]{0, 1, 1, 2, 0, 1, 0, 2, 0},
new int[]{0, 1, 1, 2, 0, 1, 2, 0, 0}, new int[]{0, 1, 1, 2, 0, 2, 0, 1, 0},
new int[]{0, 1, 1, 2, 0, 2, 1, 0, 0}, new int[]{0, 1, 1, 2, 0, 2, 2, 2, 0},
new int[]{0, 1, 1, 2, 1, 2, 0, 0, 0}, new int[]{0, 1, 1, 2, 2, 1, 0, 0, 0},
new int[]{0, 1, 2, 0, 0, 0, 1, 1, 2}, new int[]{0, 1, 2, 0, 0, 0, 1, 2, 1},
new int[]{0, 1, 2, 0, 0, 0, 2, 1, 1}, new int[]{0, 1, 2, 0, 0, 1, 0, 1, 2},
new int[]{0, 1, 2, 0, 0, 1, 0, 2, 1}, new int[]{0, 1, 2, 0, 0, 1, 1, 2, 0},
new int[]{0, 1, 2, 0, 0, 1, 2, 1, 0}, new int[]{0, 1, 2, 0, 0, 2, 0, 1, 1},
new int[]{0, 1, 2, 0, 0, 2, 1, 1, 0}, new int[]{0, 1, 2, 0, 1, 0, 0, 2, 1},
new int[]{0, 1, 2, 0, 1, 1, 0, 0, 2}, new int[]{0, 1, 2, 0, 1, 1, 0, 2, 0},
new int[]{0, 1, 2, 0, 1, 2, 0, 0, 1}, new int[]{0, 1, 2, 0, 1, 2, 0, 2, 2},
new int[]{0, 1, 2, 0, 2, 0, 0, 1, 1}, new int[]{0, 1, 2, 0, 2, 1, 0, 0, 1},
new int[]{0, 1, 2, 0, 2, 1, 0, 1, 0}, new int[]{0, 1, 2, 0, 2, 1, 0, 2, 2},
new int[]{0, 1, 2, 0, 2, 2, 0, 1, 2}, new int[]{0, 1, 2, 0, 2, 2, 0, 2, 1},
new int[]{0, 1, 2, 1, 0, 0, 1, 2, 0}, new int[]{0, 1, 2, 1, 0, 0, 2, 1, 0},
new int[]{0, 1, 2, 1, 0, 1, 0, 2, 0}, new int[]{0, 1, 2, 1, 0, 1, 2, 0, 0},
new int[]{0, 1, 2, 1, 0, 2, 0, 1, 0}, new int[]{0, 1, 2, 1, 0, 2, 1, 0, 0},
new int[]{0, 1, 2, 1, 0, 2, 2, 2, 0}, new int[]{0, 1, 2, 1, 1, 2, 0, 0, 0},
new int[]{0, 1, 2, 1, 2, 1, 0, 0, 0}, new int[]{0, 1, 2, 2, 0, 0, 1, 1, 0},
new int[]{0, 1, 2, 2, 0, 1, 0, 1, 0}, new int[]{0, 1, 2, 2, 0, 1, 1, 0, 0},
new int[]{0, 1, 2, 2, 0, 1, 2, 2, 0}, new int[]{0, 1, 2, 2, 0, 2, 1, 2, 0},
new int[]{0, 1, 2, 2, 0, 2, 2, 1, 0}, new int[]{0, 1, 2, 2, 1, 1, 0, 0, 0},
new int[]{0, 2, 0, 0, 1, 1, 0, 1, 2}, new int[]{0, 2, 0, 0, 1, 1, 0, 2, 1},
new int[]{0, 2, 0, 0, 1, 2, 0, 1, 1}, new int[]{0, 2, 0, 0, 2, 1, 0, 1, 1},
new int[]{0, 2, 0, 1, 0, 1, 0, 1, 2}, new int[]{0, 2, 0, 1, 0, 1, 0, 2, 1},
```

```
new int[]{0, 2, 0, 1, 0, 1, 1, 2, 0}, new int[]{0, 2, 0, 1, 0, 1, 2, 1, 0},
new int[]{0, 2, 0, 1, 0, 2, 0, 1, 1}, new int[]{0, 2, 0, 1, 0, 2, 1, 1, 0},
new int[]{0, 2, 0, 1, 1, 0, 1, 2, 0}, new int[]{0, 2, 0, 1, 1, 0, 2, 1, 0},
new int[]{0, 2, 0, 1, 2, 0, 1, 1, 0}, new int[]{0, 2, 0, 2, 0, 1, 0, 1, 1},
new int[]{0, 2, 0, 2, 0, 1, 1, 1, 0}, new int[]{0, 2, 0, 2, 1, 0, 1, 1, 0},
new int[]{0, 2, 1, 0, 0, 0, 1, 1, 2}, new int[]{0, 2, 1, 0, 0, 0, 1, 2, 1},
new int[]{0, 2, 1, 0, 0, 0, 2, 1, 1}, new int[]{0, 2, 1, 0, 0, 1, 0, 1, 2},
new int[]{0, 2, 1, 0, 0, 1, 1, 2, 0}, new int[]{0, 2, 1, 0, 0, 1, 2, 1, 0},
new int[]{0, 2, 1, 0, 0, 2, 0, 1, 1}, new int[]{0, 2, 1, 0, 0, 2, 1, 1, 0},
new int[]{0, 2, 1, 0, 1, 0, 0, 1, 2}, new int[]{0, 2, 1, 0, 1, 0, 0, 2, 1},
new int[]{0, 2, 1, 0, 1, 1, 0, 0, 2}, new int[]{0, 2, 1, 0, 1, 1, 0, 2, 0},
new int[]{0, 2, 1, 0, 1, 2, 0, 0, 1}, new int[]{0, 2, 1, 0, 1, 2, 0, 1, 0},
new int[]{0, 2, 1, 0, 1, 2, 0, 2, 2}, new int[]{0, 2, 1, 0, 2, 0, 0, 1, 1},
new int[]{0, 2, 1, 0, 2, 1, 0, 1, 0}, new int[]{0, 2, 1, 0, 2, 1, 0, 2, 2},
new int[]{0, 2, 1, 0, 2, 2, 0, 1, 2}, new int[]{0, 2, 1, 0, 2, 2, 0, 2, 1},
new int[]{0, 2, 1, 1, 0, 0, 1, 2, 0}, new int[]{0, 2, 1, 1, 0, 0, 2, 1, 0},
new int[]{0, 2, 1, 1, 0, 1, 0, 2, 0}, new int[]{0, 2, 1, 1, 0, 1, 2, 0, 0},
new int[]{0, 2, 1, 1, 0, 2, 0, 1, 0}, new int[]{0, 2, 1, 1, 0, 2, 1, 0, 0},
new int[]{0, 2, 1, 1, 0, 2, 2, 2, 0}, new int[]{0, 2, 1, 1, 1, 2, 0, 0, 0},
new int[]{0, 2, 1, 1, 2, 1, 0, 0, 0}, new int[]{0, 2, 1, 2, 0, 0, 1, 1, 0},
new int[]{0, 2, 1, 2, 0, 1, 0, 1, 0}, new int[]{0, 2, 1, 2, 0, 1, 1, 0, 0},
new int[]{0, 2, 1, 2, 0, 1, 2, 2, 0}, new int[]{0, 2, 1, 2, 0, 2, 1, 2, 0},
new int[]{0, 2, 1, 2, 0, 2, 2, 1, 0}, new int[]{0, 2, 1, 2, 1, 1, 0, 0, 0},
new int[]{0, 2, 2, 0, 0, 1, 0, 1, 1}, new int[]{0, 2, 2, 0, 0, 1, 1, 1, 0},
new int[]{0, 2, 2, 0, 1, 0, 0, 1, 1}, new int[]{0, 2, 2, 0, 1, 1, 0, 0, 1},
new int[]{0, 2, 2, 0, 1, 1, 0, 1, 0}, new int[]{0, 2, 2, 0, 1, 1, 0, 2, 2},
new int[]{0, 2, 2, 0, 1, 2, 0, 1, 2}, new int[]{0, 2, 2, 0, 1, 2, 0, 2, 1},
new int[]{0, 2, 2, 0, 2, 1, 0, 1, 2}, new int[]{0, 2, 2, 0, 2, 1, 0, 2, 1},
new int[]{0, 2, 2, 0, 2, 2, 0, 1, 1}, new int[]{0, 2, 2, 1, 0, 0, 1, 1, 0},
new int[]{0, 2, 2, 1, 0, 1, 0, 1, 0}, new int[]{0, 2, 2, 1, 0, 1, 1, 0, 0},
new int[]{0, 2, 2, 1, 0, 1, 2, 2, 0}, new int[]{0, 2, 2, 1, 0, 2, 1, 2, 0},
new int[]{0, 2, 2, 1, 0, 2, 2, 1, 0}, new int[]{0, 2, 2, 2, 0, 1, 1, 2, 0},
new int[]{0, 2, 2, 2, 0, 1, 2, 1, 0}, new int[]{0, 2, 2, 2, 0, 2, 1, 1, 0},
new int[]{1, 0, 0, 0, 0, 1, 0, 1, 1}, new int[]{1, 0, 0, 0, 0, 1, 1, 0, 1},
new int[]{1, 0, 0, 0, 1, 0, 1, 1, 0}, new int[]{1, 0, 0, 1, 0, 0, 0, 1, 1},
new int[]{1, 0, 0, 1, 0, 1, 0, 0, 1}, new int[]{1, 0, 0, 1, 0, 1, 0, 1, 0},
new int[]{1, 0, 0, 1, 0, 1, 0, 2, 2}, new int[]{1, 0, 0, 1, 0, 1, 2, 0, 2},
new int[]{1, 0, 0, 1, 0, 2, 0, 1, 2}, new int[]{1, 0, 0, 1, 0, 2, 0, 2, 1},
new int[]{1, 0, 0, 1, 0, 2, 2, 0, 1}, new int[]{1, 0, 0, 1, 1, 0, 0, 1, 0},
new int[]{1, 0, 0, 1, 1, 0, 2, 2, 0}, new int[]{1, 0, 0, 1, 2, 0, 2, 1, 0},
new int[]{1, 0, 0, 2, 0, 1, 0, 1, 2}, new int[]{1, 0, 0, 2, 0, 1, 0, 2, 1},
new int[]{1, 0, 0, 2, 0, 1, 1, 0, 2}, new int[]{1, 0, 0, 2, 0, 1, 2, 0, 1},
new int[]{1, 0, 0, 2, 0, 2, 0, 1, 1}, new int[]{1, 0, 0, 2, 0, 2, 1, 0, 1},
new int[]{1, 0, 0, 2, 1, 0, 1, 2, 0}, new int[]{1, 0, 0, 2, 1, 0, 2, 1, 0},
new int[]{1, 0, 0, 2, 2, 0, 1, 1, 0}, new int[]{1, 0, 1, 0, 0, 0, 0, 1, 1},
new int[]{1, 0, 1, 0, 0, 0, 1, 0, 1}, new int[]{1, 0, 1, 0, 0, 0, 1, 1, 0},
new int[]{1, 0, 1, 0, 0, 0, 1, 2, 2}, new int[]{1, 0, 1, 0, 0, 0, 2, 1, 2},
new int[]{1, 0, 1, 0, 0, 0, 2, 2, 1}, new int[]{1, 0, 1, 0, 0, 1, 1, 0, 0},
new int[]{1, 0, 1, 0, 0, 1, 2, 0, 2}, new int[]{1, 0, 1, 0, 0, 2, 1, 0, 2},
new int[]{1, 0, 1, 0, 0, 2, 2, 0, 1}, new int[]{1, 0, 1, 0, 1, 1, 0, 0, 0},
new int[]{1, 0, 1, 1, 0, 0, 0, 0, 1}, new int[]{1, 0, 1, 1, 0, 0, 2, 0, 2},
new int[]{1, 0, 1, 1, 0, 1, 0, 0, 0}, new int[]{1, 0, 1, 1, 0, 2, 0, 0, 2},
new int[]{1, 0, 1, 1, 0, 2, 2, 0, 0}, new int[]{1, 0, 1, 1, 1, 0, 0, 0, 0},
new int[]{1, 0, 1, 1, 2, 2, 0, 0, 0}, new int[]{1, 0, 1, 2, 0, 0, 1, 0, 2},
new int[]{1, 0, 1, 2, 0, 0, 2, 0, 1}, new int[]{1, 0, 1, 2, 0, 1, 0, 0, 2},
new int[]{1, 0, 1, 2, 0, 1, 2, 0, 0}, new int[]{1, 0, 1, 2, 0, 2, 0, 0, 1},
```

```
new int[]{1, 0, 1, 2, 0, 2, 1, 0, 0}, new int[]{1, 0, 1, 2, 0, 2, 2, 0, 2},
new int[]{1, 0, 1, 2, 1, 2, 0, 0, 0}, new int[]{1, 0, 1, 2, 2, 1, 0, 0, 0},
new int[]{1, 0, 2, 0, 0, 0, 1, 1, 2}, new int[]{1, 0, 2, 0, 0, 0, 1, 2, 1},
new int[]{1, 0, 2, 0, 0, 0, 2, 1, 1}, new int[]{1, 0, 2, 0, 0, 1, 1, 0, 2},
new int[]{1, 0, 2, 0, 0, 1, 2, 0, 1}, new int[]{1, 0, 2, 0, 0, 2, 1, 0, 1},
new int[]{1, 0, 2, 1, 0, 0, 2, 0, 1}, new int[]{1, 0, 2, 1, 0, 1, 0, 0, 2},
new int[]{1, 0, 2, 1, 0, 1, 2, 0, 0}, new int[]{1, 0, 2, 1, 0, 2, 0, 0, 1},
new int[]{1, 0, 2, 1, 0, 2, 2, 0, 2}, new int[]{1, 0, 2, 1, 1, 2, 0, 0, 0},
new int[]{1, 0, 2, 1, 2, 1, 0, 0, 0}, new int[]{1, 0, 2, 2, 0, 0, 1, 0, 1},
new int[]{1, 0, 2, 2, 0, 1, 0, 0, 1}, new int[]{1, 0, 2, 2, 0, 1, 1, 0, 0},
new int[]{1, 0, 2, 2, 0, 1, 2, 0, 2}, new int[]{1, 0, 2, 2, 0, 2, 1, 0, 2},
new int[]{1, 0, 2, 2, 0, 2, 2, 0, 1}, new int[]{1, 0, 2, 2, 1, 1, 0, 0, 0},
new int[]{1, 1, 0, 0, 0, 0, 0, 1, 1}, new int[]{1, 1, 0, 0, 0, 0, 1, 0, 1},
new int[]{1, 1, 0, 0, 0, 0, 1, 1, 0}, new int[]{1, 1, 0, 0, 0, 0, 1, 2, 2},
new int[]{1, 1, 0, 0, 0, 0, 2, 1, 2}, new int[]{1, 1, 0, 0, 0, 0, 2, 2, 1},
new int[]{1, 1, 0, 0, 0, 1, 0, 0, 1}, new int[]{1, 1, 0, 0, 0, 1, 0, 1, 0},
new int[]{1, 1, 0, 0, 0, 1, 0, 2, 2}, new int[]{1, 1, 0, 0, 0, 2, 0, 1, 2},
new int[]{1, 1, 0, 0, 0, 2, 0, 2, 1}, new int[]{1, 1, 0, 0, 1, 0, 1, 0, 0},
new int[]{1, 1, 0, 0, 1, 0, 2, 2, 0}, new int[]{1, 1, 0, 0, 1, 1, 0, 0, 0},
new int[]{1, 1, 0, 0, 2, 0, 1, 2, 0}, new int[]{1, 1, 0, 0, 2, 0, 2, 1, 0},
new int[]{1, 1, 0, 1, 0, 0, 0, 0, 1}, new int[]{1, 1, 0, 1, 0, 0, 0, 1, 0},
new int[]{1, 1, 0, 1, 0, 0, 0, 2, 2}, new int[]{1, 1, 0, 1, 0, 0, 2, 2, 0},
new int[]{1, 1, 0, 1, 0, 1, 0, 0, 0}, new int[]{1, 1, 0, 1, 0, 2, 0, 0, 2},
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, 2, 0, 0},
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, 0, 2, 0}, new int[]{1, 1, 0, 2, 1, 0, 2, 0, 0},
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, 2, 0},
new int[]{1, 1, 0, 2, 2, 1, 0, 0, 0}, new int[]{1, 1, 2, 0, 0, 0, 0, 1, 2},
new int[]{1, 1, 2, 0, 0, 0, 0, 2, 1}, new int[]{1, 1, 2, 0, 0, 0, 1, 0, 2},
new int[]{1, 1, 2, 0, 0, 0, 1, 2, 0}, new int[]{1, 1, 2, 0, 0, 0, 2, 0, 1},
new int[]{1, 1, 2, 0, 0, 0, 2, 1, 0}, new int[]{1, 1, 2, 0, 0, 0, 2, 2, 2},
new int[]{1, 1, 2, 0, 1, 2, 0, 0, 0}, new int[]{1, 1, 2, 0, 2, 1, 0, 0, 0},
new int[]{1, 1, 2, 1, 0, 2, 0, 0, 0}, new int[]{1, 1, 2, 1, 2, 0, 0, 0, 0},
new int[]{1, 1, 2, 2, 0, 1, 0, 0, 0}, new int[]{1, 1, 2, 2, 1, 0, 0, 0, 0},
new int[]{1, 1, 2, 2, 2, 2, 0, 0, 0}, new int[]{1, 2, 0, 0, 0, 0, 1, 1, 2},
new int[]{1, 2, 0, 0, 0, 0, 1, 2, 1}, new int[]{1, 2, 0, 0, 0, 0, 2, 1, 1},
new int[]{1, 2, 0, 0, 0, 1, 0, 1, 2}, new int[]{1, 2, 0, 0, 0, 1, 0, 2, 1},
new int[]{1, 2, 0, 0, 0, 2, 0, 1, 1}, new int[]{1, 2, 0, 0, 1, 0, 1, 2, 0},
new int[]{1, 2, 0, 0, 1, 0, 2, 1, 0}, new int[]{1, 2, 0, 0, 2, 0, 1, 1, 0},
new int[]{1, 2, 0, 1, 0, 0, 0, 1, 2}, new int[]{1, 2, 0, 1, 0, 0, 0, 2, 1},
new int[]{1, 2, 0, 1, 0, 0, 2, 1, 0}, new int[]{1, 2, 0, 1, 0, 1, 0, 0, 2},
new int[]{1, 2, 0, 1, 0, 1, 0, 2, 0}, new int[]{1, 2, 0, 1, 0, 2, 0, 0, 1},
new int[]{1, 2, 0, 1, 0, 2, 0, 1, 0}, new int[]{1, 2, 0, 1, 0, 2, 0, 2, 2},
new int[]{1, 2, 0, 1, 1, 0, 0, 2, 0}, new int[]{1, 2, 0, 1, 1, 0, 2, 0, 0},
new int[]{1, 2, 0, 1, 1, 2, 0, 0, 0}, new int[]{1, 2, 0, 1, 2, 0, 0, 1, 0},
new int[]{1, 2, 0, 1, 2, 0, 2, 2, 0}, new int[]{1, 2, 0, 1, 2, 1, 0, 0, 0},
new int[]{1, 2, 0, 2, 0, 0, 0, 1, 1}, new int[]{1, 2, 0, 2, 0, 0, 1, 1, 0},
new int[]{1, 2, 0, 2, 0, 1, 0, 0, 1}, new int[]{1, 2, 0, 2, 0, 1, 0, 1, 0},
new int[]{1, 2, 0, 2, 0, 1, 0, 2, 2}, new int[]{1, 2, 0, 2, 0, 2, 0, 1, 2},
new int[]{1, 2, 0, 2, 0, 2, 0, 2, 1}, new int[]{1, 2, 0, 2, 1, 0, 0, 1, 0},
```

```
new int[]{1, 2, 0, 2, 1, 0, 1, 0, 0}, new int[]{1, 2, 0, 2, 1, 0, 2, 2, 0},
new int[]{1, 2, 0, 2, 1, 1, 0, 0, 0}, new int[]{1, 2, 0, 2, 2, 0, 1, 2, 0},
new int[]{1, 2, 0, 2, 2, 0, 2, 1, 0}, new int[]{1, 2, 1, 0, 0, 0, 0, 1, 2},
new int[]{1, 2, 1, 0, 0, 0, 0, 2, 1}, new int[]{1, 2, 1, 0, 0, 0, 1, 0, 2},
new int[]{1, 2, 1, 0, 0, 0, 1, 2, 0}, new int[]{1, 2, 1, 0, 0, 0, 2, 0, 1},
new int[]{1, 2, 1, 0, 0, 0, 2, 1, 0}, new int[]{1, 2, 1, 0, 0, 0, 2, 2, 2},
new int[]{1, 2, 1, 0, 1, 2, 0, 0, 0}, new int[]{1, 2, 1, 0, 2, 1, 0, 0, 0},
new int[]{1, 2, 1, 1, 0, 2, 0, 0, 0}, new int[]{1, 2, 1, 1, 2, 0, 0, 0, 0},
new int[]{1, 2, 1, 2, 0, 1, 0, 0, 0}, new int[]{1, 2, 1, 2, 1, 0, 0, 0, 0},
new int[]{1, 2, 1, 2, 2, 2, 0, 0, 0}, new int[]{1, 2, 2, 0, 0, 0, 0, 1, 1},
new int[]{1, 2, 2, 0, 0, 0, 1, 0, 1}, new int[]{1, 2, 2, 0, 0, 0, 1, 1, 0},
new int[]{1, 2, 2, 0, 0, 0, 1, 2, 2}, new int[]{1, 2, 2, 0, 0, 0, 2, 1, 2},
new int[]{1, 2, 2, 0, 0, 0, 2, 2, 1}, new int[]{1, 2, 2, 0, 1, 1, 0, 0, 0},
new int[]{1, 2, 2, 1, 0, 1, 0, 0, 0}, new int[]{1, 2, 2, 1, 1, 0, 0, 0, 0},
new int[]{1, 2, 2, 1, 2, 2, 0, 0, 0}, new int[]{1, 2, 2, 2, 1, 2, 0, 0, 0},
new int[]{1, 2, 2, 2, 2, 1, 0, 0, 0}, new int[]{2, 0, 0, 1, 0, 1, 0, 1, 2},
new int[]{2, 0, 0, 1, 0, 1, 0, 2, 1}, new int[]{2, 0, 0, 1, 0, 1, 1, 0, 2},
new int[]{2, 0, 0, 1, 0, 1, 2, 0, 1}, new int[]{2, 0, 0, 1, 0, 2, 0, 1, 1},
new int[]{2, 0, 0, 1, 0, 2, 1, 0, 1}, new int[]{2, 0, 0, 1, 1, 0, 1, 2, 0},
new int[]{2, 0, 0, 1, 1, 0, 2, 1, 0}, new int[]{2, 0, 0, 1, 2, 0, 1, 1, 0},
new int[]{2, 0, 0, 2, 0, 1, 0, 1, 1}, new int[]{2, 0, 0, 2, 0, 1, 1, 0, 1},
new int[]{2, 0, 0, 2, 1, 0, 1, 1, 0}, new int[]{2, 0, 1, 0, 0, 0, 1, 1, 2},
new int[]{2, 0, 1, 0, 0, 0, 1, 2, 1}, new int[]{2, 0, 1, 0, 0, 0, 2, 1, 1},
new int[]{2, 0, 1, 0, 0, 1, 1, 0, 2}, new int[]{2, 0, 1, 0, 0, 2, 1, 0, 1},
new int[]{2, 0, 1, 1, 0, 0, 1, 0, 2}, new int[]{2, 0, 1, 1, 0, 0, 2, 0, 1},
new int[]{2, 0, 1, 1, 0, 1, 0, 0, 2}, new int[]{2, 0, 1, 1, 0, 1, 2, 0, 0},
new int[]{2, 0, 1, 1, 0, 2, 0, 0, 1}, new int[]{2, 0, 1, 1, 0, 2, 1, 0, 0},
new int[]{2, 0, 1, 1, 0, 2, 2, 0, 2}, new int[]{2, 0, 1, 1, 1, 2, 0, 0, 0},
new int[]{2, 0, 1, 1, 2, 1, 0, 0, 0}, new int[]{2, 0, 1, 2, 0, 0, 1, 0, 1},
new int[]{2, 0, 1, 2, 0, 1, 1, 0, 0}, new int[]{2, 0, 1, 2, 0, 1, 2, 0, 2},
new int[]{2, 0, 1, 2, 0, 2, 1, 0, 2}, new int[]{2, 0, 1, 2, 0, 2, 2, 0, 1},
new int[]{2, 0, 1, 2, 1, 1, 0, 0, 0}, new int[]{2, 0, 2, 0, 0, 1, 1, 0, 1},
new int[]{2, 0, 2, 1, 0, 0, 1, 0, 1}, new int[]{2, 0, 2, 1, 0, 1, 0, 0, 1},
new int[]{2, 0, 2, 1, 0, 1, 1, 0, 0}, new int[]{2, 0, 2, 1, 0, 1, 2, 0, 2},
new int[]{2, 0, 2, 1, 0, 2, 1, 0, 2}, new int[]{2, 0, 2, 1, 0, 2, 2, 0, 1},
new int[]{2, 0, 2, 2, 0, 1, 1, 0, 2}, new int[]{2, 0, 2, 2, 0, 1, 2, 0, 1},
new int[]{2, 0, 2, 2, 0, 2, 1, 0, 1}, new int[]{2, 1, 0, 0, 0, 0, 1, 1, 2},
new int[]{2, 1, 0, 0, 0, 0, 1, 2, 1}, new int[]{2, 1, 0, 0, 0, 0, 2, 1, 1},
new int[]{2, 1, 0, 0, 0, 1, 0, 1, 2}, new int[]{2, 1, 0, 0, 0, 1, 0, 2, 1},
new int[]{2, 1, 0, 0, 0, 2, 0, 1, 1}, new int[]{2, 1, 0, 0, 1, 0, 1, 2, 0},
new int[]{2, 1, 0, 0, 2, 0, 1, 1, 0}, new int[]{2, 1, 0, 1, 0, 0, 0, 1, 2},
new int[]{2, 1, 0, 1, 0, 0, 0, 2, 1}, new int[]{2, 1, 0, 1, 0, 0, 1, 2, 0},
new int[]{2, 1, 0, 1, 0, 0, 2, 1, 0}, new int[]{2, 1, 0, 1, 0, 1, 0, 0, 2},
new int[]{2, 1, 0, 1, 0, 1, 0, 2, 0}, new int[]{2, 1, 0, 1, 0, 2, 0, 0, 1},
new int[]{2, 1, 0, 1, 0, 2, 0, 1, 0}, new int[]{2, 1, 0, 1, 0, 2, 0, 2, 2},
new int[]{2, 1, 0, 1, 1, 0, 0, 2, 0}, new int[]{2, 1, 0, 1, 1, 0, 2, 0, 0},
new int[]{2, 1, 0, 1, 1, 2, 0, 0, 0}, new int[]{2, 1, 0, 1, 2, 0, 0, 1, 0},
new int[]{2, 1, 0, 1, 2, 0, 1, 0, 0}, new int[]{2, 1, 0, 1, 2, 0, 2, 2, 0},
new int[]{2, 1, 0, 1, 2, 1, 0, 0, 0}, new int[]{2, 1, 0, 2, 0, 0, 0, 1, 1},
new int[]{2, 1, 0, 2, 0, 0, 1, 1, 0}, new int[]{2, 1, 0, 2, 0, 1, 0, 0, 1},
new int[]{2, 1, 0, 2, 0, 1, 0, 1, 0}, new int[]{2, 1, 0, 2, 0, 1, 0, 2, 2},
new int[]{2, 1, 0, 2, 0, 2, 0, 1, 2}, new int[]{2, 1, 0, 2, 0, 2, 0, 2, 1},
new int[]{2, 1, 0, 2, 1, 0, 1, 0, 0}, new int[]{2, 1, 0, 2, 1, 0, 2, 2, 0},
new int[]{2, 1, 0, 2, 1, 1, 0, 0, 0}, new int[]{2, 1, 0, 2, 2, 0, 1, 2, 0},
new int[]{2, 1, 0, 2, 2, 0, 2, 1, 0}, new int[]{2, 1, 1, 0, 0, 0, 0, 1, 2},
new int[]{2, 1, 1, 0, 0, 0, 0, 2, 1}, new int[]{2, 1, 1, 0, 0, 0, 1, 0, 2},
```

```
new int[]{2, 1, 1, 0, 0, 0, 1, 2, 0}, new int[]{2, 1, 1, 0, 0, 0, 2, 0, 1},
new int[]{2, 1, 1, 0, 0, 0, 2, 1, 0}, new int[]{2, 1, 1, 0, 0, 0, 2, 2, 2},
new int[]{2, 1, 1, 0, 1, 2, 0, 0, 0}, new int[]{2, 1, 1, 0, 2, 1, 0, 0, 0},
new int[]{2, 1, 1, 1, 0, 2, 0, 0, 0}, new int[]{2, 1, 1, 1, 2, 0, 0, 0, 0},
new int[]{2, 1, 1, 2, 0, 1, 0, 0, 0}, new int[]{2, 1, 1, 2, 1, 0, 0, 0, 0},
new int[]{2, 1, 1, 2, 2, 2, 0, 0, 0}, new int[]{2, 1, 2, 0, 0, 0, 0, 1, 1},
new int[]{2, 1, 2, 0, 0, 0, 1, 0, 1}, new int[]{2, 1, 2, 0, 0, 0, 1, 1, 0},
new int[]{2, 1, 2, 0, 0, 0, 1, 2, 2}, new int[]{2, 1, 2, 0, 0, 0, 2, 1, 2},
new int[]{2, 1, 2, 0, 0, 0, 2, 2, 1}, new int[]{2, 1, 2, 0, 1, 1, 0, 0, 0},
new int[]{2, 1, 2, 1, 0, 1, 0, 0, 0}, new int[]{2, 1, 2, 1, 1, 0, 0, 0, 0},
new int[]{2, 1, 2, 1, 2, 2, 0, 0, 0}, new int[]{2, 1, 2, 2, 1, 2, 0, 0, 0},
new int[]{2, 1, 2, 2, 2, 1, 0, 0, 0}, new int[]{2, 2, 0, 0, 0, 1, 0, 1, 1},
new int[]{2, 2, 0, 0, 1, 0, 1, 1, 0}, new int[]{2, 2, 0, 1, 0, 0, 0, 1, 1},
new int[]{2, 2, 0, 1, 0, 0, 1, 1, 0}, new int[]{2, 2, 0, 1, 0, 1, 0, 0, 1},
new int[]{2, 2, 0, 1, 0, 1, 0, 1, 0}, new int[]{2, 2, 0, 1, 0, 1, 0, 2, 2},
new int[]{2, 2, 0, 1, 0, 2, 0, 1, 2}, new int[]{2, 2, 0, 1, 0, 2, 0, 2, 1},
new int[]{2, 2, 0, 1, 1, 0, 0, 1, 0}, new int[]{2, 2, 0, 1, 1, 0, 1, 0, 0},
new int[]{2, 2, 0, 1, 1, 0, 2, 2, 0}, new int[]{2, 2, 0, 1, 2, 0, 1, 2, 0},
new int[]{2, 2, 0, 1, 2, 0, 2, 1, 0}, new int[]{2, 2, 0, 2, 0, 1, 0, 1, 2},
new int[]{2, 2, 0, 2, 0, 1, 0, 2, 1}, new int[]{2, 2, 0, 2, 0, 2, 0, 1, 1},
new int[]{2, 2, 0, 2, 1, 0, 1, 2, 0}, new int[]{2, 2, 0, 2, 1, 0, 2, 1, 0},
new int[]{2, 2, 0, 2, 2, 0, 1, 1, 0}, new int[]{2, 2, 1, 0, 0, 0, 0, 1, 1},
new int[]{2, 2, 1, 0, 0, 0, 1, 0, 1}, new int[]{2, 2, 1, 0, 0, 0, 1, 1, 0},
new int[]{2, 2, 1, 0, 0, 0, 1, 2, 2}, new int[]{2, 2, 1, 0, 0, 0, 2, 1, 2},
new int[]{2, 2, 1, 0, 0, 0, 2, 2, 1}, new int[]{2, 2, 1, 0, 1, 1, 0, 0, 0},
new int[]{2, 2, 1, 1, 0, 1, 0, 0, 0}, new int[]{2, 2, 1, 1, 1, 0, 0, 0, 0},
new int[]{2, 2, 1, 1, 2, 2, 0, 0, 0}, new int[]{2, 2, 1, 2, 1, 2, 0, 0, 0},
new int[]{2, 2, 1, 2, 2, 1, 0, 0, 0}, new int[]{2, 2, 2, 0, 0, 0, 1, 1, 2},
new int[]{2, 2, 2, 0, 0, 0, 1, 2, 1}, new int[]{2, 2, 2, 0, 0, 0, 2, 1, 1},
new int[]{2, 2, 2, 1, 1, 2, 0, 0, 0}, new int[]{2, 2, 2, 1, 2, 1, 0, 0, 0},
new int[]{2, 2, 2, 2, 1, 1, 0, 0, 0}, new int[]{0, 0, 1, 0, 0, 1, 1, 2, 1},
new int[]{0, 0, 1, 0, 0, 1, 2, 1, 1}, new int[]{0, 0, 1, 0, 0, 2, 1, 1, 1},
new int[]{0, 0, 1, 0, 1, 0, 1, 1, 2}, new int[]{0, 0, 1, 0, 1, 0, 1, 2, 1},
new int[]{0, 0, 1, 0, 1, 1, 1, 0, 2}, new int[]{0, 0, 1, 0, 1, 1, 1, 2, 0},
new int[]{0, 0, 1, 0, 1, 1, 2, 0, 1}, new int[]{0, 0, 1, 0, 1, 2, 1, 0, 1},
new int[]{0, 0, 1, 0, 1, 2, 1, 1, 0}, new int[]{0, 0, 1, 0, 1, 2, 1, 2, 2},
new int[]{0, 0, 1, 0, 2, 0, 1, 1, 1}, new int[]{0, 0, 1, 0, 2, 1, 1, 0, 1},
new int[]{0, 0, 1, 0, 2, 1, 2, 2, 1}, new int[]{0, 0, 1, 1, 0, 1, 0, 2, 1},
new int[]{0, 0, 1, 1, 1, 0, 1, 0, 2}, new int[]{0, 0, 1, 1, 1, 0, 1, 2, 0},
new int[]{0, 0, 1, 1, 1, 1, 0, 0, 2}, new int[]{0, 0, 1, 1, 1, 1, 0, 2, 0},
new int[]{0, 0, 1, 1, 1, 1, 2, 0, 0}, new int[]{0, 0, 1, 1, 1, 2, 1, 0, 0},
new int[]{0, 0, 1, 1, 2, 1, 0, 0, 1}, new int[]{0, 0, 1, 2, 0, 0, 1, 1, 1},
new int[]{0, 0, 1, 2, 0, 1, 0, 1, 1}, new int[]{0, 0, 1, 2, 0, 1, 2, 2, 1},
new int[]{0, 0, 1, 2, 1, 0, 1, 0, 1}, new int[]{0, 0, 1, 2, 1, 0, 1, 1, 0},
new int[]{0, 0, 1, 2, 1, 0, 1, 2, 2}, new int[]{0, 0, 1, 2, 1, 1, 0, 0, 1},
new int[]{0, 0, 1, 2, 1, 1, 1, 0, 0}, new int[]{0, 0, 1, 2, 1, 2, 1, 0, 2},
new int[]{0, 0, 1, 2, 1, 2, 1, 2, 0}, new int[]{0, 0, 1, 2, 2, 1, 0, 2, 1},
new int[]{0, 0, 1, 2, 2, 1, 2, 0, 1}, new int[]{0, 0, 2, 0, 0, 1, 1, 1, 1},
new int[]{0, 0, 2, 0, 1, 0, 1, 1, 1}, new int[]{0, 0, 2, 0, 2, 2, 1, 1, 1},
new int[]{0, 0, 2, 1, 0, 0, 1, 1, 1}, new int[]{0, 0, 2, 1, 1, 1, 0, 0, 1},
new int[]{0, 0, 2, 1, 1, 1, 0, 1, 0}, new int[]{0, 0, 2, 1, 1, 1, 0, 2, 2},
new int[]{0, 0, 2, 1, 1, 1, 1, 0, 0}, new int[]{0, 0, 2, 1, 1, 1, 2, 0, 2},
new int[]{0, 0, 2, 1, 1, 1, 2, 2, 0}, new int[]{0, 0, 2, 2, 0, 2, 1, 1, 1},
new int[]{0, 0, 2, 2, 2, 0, 1, 1, 1}, new int[]{0, 1, 0, 0, 0, 2, 1, 1, 1},
new int[]{0, 1, 0, 0, 1, 0, 1, 1, 2}, new int[]{0, 1, 0, 0, 1, 0, 2, 1, 1},
new int[]{0, 1, 0, 0, 1, 1, 2, 1, 0}, new int[]{0, 1, 0, 0, 1, 2, 1, 1, 0},
```

```
new int[]{0, 1, 0, 0, 1, 2, 2, 1, 2}, new int[]{0, 1, 0, 0, 2, 0, 1, 1, 1},
new int[]{0, 1, 0, 1, 1, 0, 0, 1, 2}, new int[]{0, 1, 0, 1, 1, 1, 0, 0, 2},
new int[]{0, 1, 0, 1, 1, 1, 0, 2, 0}, new int[]{0, 1, 0, 1, 1, 1, 2, 0, 0},
new int[]{0, 1, 0, 1, 1, 2, 0, 1, 0}, new int[]{0, 1, 0, 2, 0, 0, 1, 1, 1},
new int[]{0, 1, 0, 2, 1, 0, 0, 1, 1}, new int[]{0, 1, 0, 2, 1, 0, 2, 1, 2},
new int[]{0, 1, 0, 2, 1, 1, 0, 1, 0}, new int[]{0, 1, 0, 2, 1, 2, 0, 1, 2},
new int[]{0, 1, 0, 2, 1, 2, 2, 1, 0}, new int[]{0, 1, 1, 0, 0, 1, 2, 0, 1},
new int[]{0, 1, 1, 0, 1, 0, 1, 0, 2}, new int[]{0, 1, 1, 0, 1, 0, 1, 2, 0},
new int[]{0, 1, 1, 0, 1, 0, 2, 1, 0}, new int[]{0, 1, 1, 0, 1, 2, 1, 0, 0},
new int[]{0, 1, 1, 2, 0, 1, 0, 0, 1}, new int[]{0, 1, 1, 2, 1, 0, 0, 1, 0},
new int[]{0, 1, 1, 2, 1, 0, 1, 0, 0}, new int[]{0, 1, 2, 0, 1, 0, 1, 1, 0},
new int[]{0, 1, 2, 0, 1, 0, 2, 1, 2}, new int[]{0, 1, 2, 0, 1, 2, 2, 1, 0},
new int[]{0, 1, 2, 1, 1, 0, 0, 1, 0}, new int[]{0, 1, 2, 2, 1, 0, 0, 1, 2},
new int[]{0, 1, 2, 2, 1, 0, 2, 1, 0}, new int[]{0, 1, 2, 2, 1, 2, 0, 1, 0},
new int[]{0, 2, 0, 0, 0, 1, 1, 1, 1}, new int[]{0, 2, 0, 0, 1, 0, 1, 1, 1},
new int[]{0, 2, 0, 0, 2, 2, 1, 1, 1}, new int[]{0, 2, 0, 1, 0, 0, 1, 1, 1},
new int[]{0, 2, 0, 1, 1, 1, 0, 0, 1}, new int[]{0, 2, 0, 1, 1, 1, 0, 1, 0},
new int[]{0, 2, 0, 1, 1, 1, 0, 2, 2}, new int[]{0, 2, 0, 1, 1, 1, 1, 0, 0},
new int[]{0, 2, 0, 1, 1, 1, 2, 0, 2}, new int[]{0, 2, 0, 1, 1, 1, 2, 2, 0},
new int[]{0, 2, 0, 2, 0, 2, 1, 1, 1}, new int[]{0, 2, 0, 2, 2, 0, 1, 1, 1},
new int[]{0, 2, 1, 0, 0, 1, 1, 0, 1}, new int[]{0, 2, 1, 0, 0, 1, 2, 2, 1},
new int[]{0, 2, 1, 0, 1, 0, 1, 0, 1}, new int[]{0, 2, 1, 0, 1, 0, 1, 1, 0},
new int[]{0, 2, 1, 0, 1, 0, 1, 2, 2}, new int[]{0, 2, 1, 0, 1, 1, 1, 0, 0},
new int[]{0, 2, 1, 0, 1, 2, 1, 0, 2}, new int[]{0, 2, 1, 0, 1, 2, 1, 2, 0},
new int[]{0, 2, 1, 0, 2, 1, 2, 0, 1}, new int[]{0, 2, 1, 1, 0, 1, 0, 0, 1},
new int[]{0, 2, 1, 1, 1, 0, 1, 0, 0}, new int[]{0, 2, 1, 2, 0, 1, 0, 2, 1},
new int[]{0, 2, 1, 2, 0, 1, 2, 0, 1}, new int[]{0, 2, 1, 2, 1, 0, 1, 0, 2},
new int[]{0, 2, 1, 2, 1, 0, 1, 2, 0}, new int[]{0, 2, 1, 2, 1, 2, 1, 0, 0},
new int[]{0, 2, 1, 2, 2, 1, 0, 0, 1}, new int[]{0, 2, 2, 0, 0, 2, 1, 1, 1},
new int[]{0, 2, 2, 0, 2, 0, 1, 1, 1}, new int[]{0, 2, 2, 1, 1, 1, 0, 0, 2},
new int[]{0, 2, 2, 1, 1, 1, 0, 2, 0}, new int[]{0, 2, 2, 1, 1, 1, 2, 0, 0},
new int[]{0, 2, 2, 2, 0, 0, 1, 1, 1}, new int[]{1, 0, 0, 0, 0, 2, 1, 1, 1},
new int[]{1, 0, 0, 0, 1, 0, 1, 2, 1}, new int[]{1, 0, 0, 0, 1, 0, 2, 1, 1},
new int[]{1, 0, 0, 0, 1, 1, 0, 2, 1}, new int[]{1, 0, 0, 0, 1, 1, 2, 0, 1},
new int[]{1, 0, 0, 0, 1, 2, 0, 1, 1}, new int[]{1, 0, 0, 0, 1, 2, 1, 0, 1},
new int[]{1, 0, 0, 0, 1, 2, 2, 2, 1}, new int[]{1, 0, 0, 0, 2, 0, 1, 1, 1},
new int[]{1, 0, 0, 1, 0, 0, 1, 1, 2}, new int[]{1, 0, 0, 1, 0, 0, 1, 2, 1},
new int[]{1, 0, 0, 1, 0, 1, 1, 2, 0}, new int[]{1, 0, 0, 1, 0, 2, 1, 1, 0},
new int[]{1, 0, 0, 1, 0, 2, 1, 2, 2}, new int[]{1, 0, 0, 1, 1, 0, 0, 2, 1},
new int[]{1, 0, 0, 1, 1, 0, 1, 0, 2}, new int[]{1, 0, 0, 1, 1, 0, 2, 0, 1},
new int[]{1, 0, 0, 1, 1, 1, 0, 0, 2}, new int[]{1, 0, 0, 1, 1, 1, 0, 2, 0},
new int[]{1, 0, 0, 1, 1, 1, 2, 0, 0}, new int[]{1, 0, 0, 1, 1, 2, 0, 0, 1},
new int[]{1, 0, 0, 1, 1, 2, 1, 0, 0}, new int[]{1, 0, 0, 1, 2, 0, 1, 0, 1},
new int[]{1, 0, 0, 1, 2, 0, 1, 2, 2}, new int[]{1, 0, 0, 1, 2, 1, 1, 0, 0},
new int[]{1, 0, 0, 1, 2, 2, 1, 0, 2}, new int[]{1, 0, 0, 1, 2, 2, 1, 2, 0},
new int[]{1, 0, 0, 2, 0, 0, 1, 1, 1}, new int[]{1, 0, 0, 2, 1, 0, 0, 1, 1},
new int[]{1, 0, 0, 2, 1, 0, 1, 0, 1}, new int[]{1, 0, 0, 2, 1, 0, 2, 2, 1},
new int[]{1, 0, 0, 2, 1, 1, 0, 0, 1}, new int[]{1, 0, 0, 2, 1, 2, 0, 2, 1},
new int[]{1, 0, 0, 2, 1, 2, 2, 0, 1}, new int[]{1, 0, 1, 0, 0, 1, 0, 2, 1},
new int[]{1, 0, 1, 0, 1, 0, 0, 2, 1}, new int[]{1, 0, 1, 0, 1, 0, 1, 0, 2},
new int[]{1, 0, 1, 0, 1, 0, 1, 2, 0}, new int[]{1, 0, 1, 0, 1, 0, 2, 0, 1},
new int[]{1, 0, 1, 0, 1, 2, 0, 0, 1}, new int[]{1, 0, 1, 0, 1, 2, 1, 0, 0},
new int[]{1, 0, 1, 0, 2, 1, 0, 0, 1}, new int[]{1, 0, 1, 1, 0, 0, 1, 2, 0},
new int[]{1, 0, 1, 1, 2, 0, 1, 0, 0}, new int[]{1, 0, 1, 2, 1, 0, 0, 0, 1},
new int[]{1, 0, 1, 2, 1, 0, 1, 0, 0}, new int[]{1, 0, 2, 0, 1, 0, 0, 1, 1},
new int[]{1, 0, 2, 0, 1, 0, 1, 0, 1}, new int[]{1, 0, 2, 0, 1, 0, 2, 2, 1},
```

```
new int[]{1, 0, 2, 0, 1, 1, 0, 0, 1}, new int[]{1, 0, 2, 0, 1, 2, 0, 2, 1},
new int[]{1, 0, 2, 0, 1, 2, 2, 0, 1}, new int[]{1, 0, 2, 1, 0, 0, 1, 1, 0},
new int[]{1, 0, 2, 1, 0, 0, 1, 2, 2}, new int[]{1, 0, 2, 1, 0, 2, 1, 2, 0},
new int[]{1, 0, 2, 1, 1, 0, 0, 0, 1}, new int[]{1, 0, 2, 1, 1, 0, 1, 0, 0},
new int[]{1, 0, 2, 1, 2, 0, 1, 0, 2}, new int[]{1, 0, 2, 1, 2, 0, 1, 2, 0},
new int[]{1, 0, 2, 1, 2, 2, 1, 0, 0}, new int[]{1, 0, 2, 2, 1, 0, 0, 2, 1},
new int[]{1, 0, 2, 2, 1, 0, 2, 0, 1}, new int[]{1, 0, 2, 2, 1, 2, 0, 0, 1},
new int[]{1, 1, 0, 0, 1, 0, 0, 1, 2}, new int[]{1, 1, 0, 0, 1, 0, 0, 2, 1},
new int[]{1, 1, 0, 0, 1, 0, 2, 0, 1}, new int[]{1, 1, 0, 0, 1, 2, 0, 0, 1},
new int[]{1, 1, 0, 0, 1, 2, 0, 1, 0}, new int[]{1, 1, 0, 1, 0, 0, 1, 0, 2},
new int[]{1, 1, 0, 1, 0, 2, 1, 0, 0}, new int[]{1, 1, 0, 2, 1, 0, 0, 0, 1},
new int[]{1, 1, 1, 0, 0, 1, 0, 0, 2}, new int[]{1, 1, 1, 0, 0, 1, 0, 2, 0},
new int[]{1, 1, 1, 0, 0, 1, 2, 0, 0}, new int[]{1, 1, 1, 0, 0, 2, 0, 0, 1},
new int[]{1, 1, 1, 0, 0, 2, 0, 1, 0}, new int[]{1, 1, 1, 0, 0, 2, 0, 2, 2},
new int[]{1, 1, 1, 0, 0, 2, 1, 0, 0}, new int[]{1, 1, 1, 0, 0, 2, 2, 0, 2},
new int[]{1, 1, 1, 0, 0, 2, 2, 2, 0}, new int[]{1, 1, 1, 0, 1, 0, 0, 0, 2},
new int[]{1, 1, 1, 0, 1, 0, 0, 2, 0}, new int[]{1, 1, 1, 0, 1, 0, 2, 0, 0},
new int[]{1, 1, 1, 0, 2, 0, 0, 0, 1}, new int[]{1, 1, 1, 0, 2, 0, 0, 1, 0},
new int[]{1, 1, 1, 0, 2, 0, 0, 2, 2}, new int[]{1, 1, 1, 0, 2, 0, 1, 0, 0},
new int[]{1, 1, 1, 0, 2, 0, 2, 0, 2}, new int[]{1, 1, 1, 0, 2, 0, 2, 2, 0},
new int[]{1, 1, 1, 0, 2, 2, 0, 0, 2}, new int[]{1, 1, 1, 0, 2, 2, 0, 2, 0},
new int[]{1, 1, 1, 0, 2, 2, 2, 0, 0}, new int[]{1, 1, 1, 1, 0, 0, 0, 0, 2},
new int[]{1, 1, 1, 1, 0, 0, 0, 2, 0}, new int[]{1, 1, 1, 1, 0, 0, 2, 0, 0},
new int[]{1, 1, 1, 2, 0, 0, 0, 0, 1}, new int[]{1, 1, 1, 2, 0, 0, 0, 1, 0},
new int[]{1, 1, 1, 2, 0, 0, 0, 2, 2}, new int[]{1, 1, 1, 2, 0, 0, 1, 0, 0},
new int[]{1, 1, 1, 2, 0, 0, 2, 0, 2}, new int[]{1, 1, 1, 2, 0, 0, 2, 2, 0},
new int[]{1, 1, 1, 2, 0, 2, 0, 0, 2}, new int[]{1, 1, 1, 2, 0, 2, 0, 2, 0},
new int[]{1, 1, 1, 2, 0, 2, 2, 0, 0}, new int[]{1, 1, 1, 2, 2, 0, 0, 0, 2},
new int[]{1, 1, 1, 2, 2, 0, 0, 2, 0}, new int[]{1, 1, 1, 2, 2, 0, 2, 0, 0},
new int[]{1, 1, 2, 0, 1, 0, 0, 0, 1}, new int[]{1, 1, 2, 0, 1, 0, 0, 1, 0},
new int[]{1, 1, 2, 1, 0, 0, 1, 0, 0}, new int[]{1, 2, 0, 0, 1, 0, 0, 1, 1},
new int[]{1, 2, 0, 0, 1, 0, 1, 0, 1}, new int[]{1, 2, 0, 0, 1, 0, 2, 2, 1},
new int[]{1, 2, 0, 0, 1, 1, 0, 0, 1}, new int[]{1, 2, 0, 0, 1, 2, 0, 2, 1},
new int[]{1, 2, 0, 0, 1, 2, 2, 0, 1}, new int[]{1, 2, 0, 1, 0, 0, 1, 0, 1},
new int[]{1, 2, 0, 1, 0, 0, 1, 2, 2}, new int[]{1, 2, 0, 1, 0, 1, 1, 0, 0},
new int[]{1, 2, 0, 1, 0, 2, 1, 0, 2}, new int[]{1, 2, 0, 1, 0, 2, 1, 2, 0},
new int[]{1, 2, 0, 1, 1, 0, 0, 0, 1}, new int[]{1, 2, 0, 1, 2, 0, 1, 0, 2},
new int[]{1, 2, 0, 1, 2, 2, 1, 0, 0}, new int[]{1, 2, 0, 2, 1, 0, 0, 2, 1},
new int[]{1, 2, 0, 2, 1, 0, 2, 0, 1}, new int[]{1, 2, 0, 2, 1, 2, 0, 0, 1},
new int[]{1, 2, 1, 0, 0, 1, 0, 0, 1}, new int[]{1, 2, 1, 0, 1, 0, 0, 0, 1},
new int[]{1, 2, 1, 0, 1, 0, 1, 0, 0}, new int[]{1, 2, 1, 1, 0, 0, 1, 0, 0},
new int[]{1, 2, 2, 0, 1, 0, 0, 2, 1}, new int[]{1, 2, 2, 0, 1, 0, 2, 0, 1},
new int[]{1, 2, 2, 0, 1, 2, 0, 0, 1}, new int[]{1, 2, 2, 1, 0, 0, 1, 0, 2},
new int[]{1, 2, 2, 1, 0, 0, 1, 2, 0}, new int[]{1, 2, 2, 1, 0, 2, 1, 0, 0},
new int[]{1, 2, 2, 1, 2, 0, 1, 0, 0}, new int[]{1, 2, 2, 2, 1, 0, 0, 0, 1},
new int[]{2, 0, 0, 0, 0, 1, 1, 1, 1}, new int[]{2, 0, 0, 0, 1, 0, 1, 1, 1},
new int[]{2, 0, 0, 0, 2, 2, 1, 1, 1}, new int[]{2, 0, 0, 1, 0, 0, 1, 1, 1},
new int[]{2, 0, 0, 1, 1, 1, 0, 0, 1}, new int[]{2, 0, 0, 1, 1, 1, 0, 1, 0},
new int[]{2, 0, 0, 1, 1, 1, 0, 2, 2}, new int[]{2, 0, 0, 1, 1, 1, 1, 0, 0},
new int[]{2, 0, 0, 1, 1, 1, 2, 0, 2}, new int[]{2, 0, 0, 1, 1, 1, 2, 2, 0},
new int[]{2, 0, 0, 2, 0, 2, 1, 1, 1}, new int[]{2, 0, 0, 2, 2, 0, 1, 1, 1},
new int[]{2, 0, 1, 0, 0, 1, 0, 1, 1}, new int[]{2, 0, 1, 0, 0, 1, 2, 2, 1},
new int[]{2, 0, 1, 0, 1, 0, 1, 0, 1}, new int[]{2, 0, 1, 0, 1, 0, 1, 1, 0},
new int[]{2, 0, 1, 0, 1, 0, 1, 2, 2}, new int[]{2, 0, 1, 0, 1, 1, 0, 0, 1},
new int[]{2, 0, 1, 0, 1, 1, 1, 0, 0}, new int[]{2, 0, 1, 0, 1, 2, 1, 0, 2},
new int[]{2, 0, 1, 0, 1, 2, 1, 2, 0}, new int[]{2, 0, 1, 0, 2, 1, 0, 2, 1},
```

```
    new int[]{2, 0, 1, 0, 2, 1, 2, 0, 1}, new int[]{2, 0, 1, 1, 1, 0, 1, 0, 0},
    new int[]{2, 0, 1, 2, 0, 1, 0, 2, 1}, new int[]{2, 0, 1, 2, 1, 0, 1, 0, 2},
    new int[]{2, 0, 1, 2, 1, 0, 1, 2, 0}, new int[]{2, 0, 1, 2, 1, 2, 1, 0, 0},
    new int[]{2, 0, 1, 2, 2, 1, 0, 0, 1}, new int[]{2, 0, 2, 0, 0, 2, 1, 1, 1},
    new int[]{2, 0, 2, 0, 2, 0, 1, 1, 1}, new int[]{2, 0, 2, 1, 1, 1, 0, 0, 2},
    new int[]{2, 0, 2, 1, 1, 1, 0, 2, 0}, new int[]{2, 0, 2, 1, 1, 1, 2, 0, 0},
    new int[]{2, 0, 2, 2, 0, 0, 1, 1, 1}, new int[]{2, 1, 0, 0, 1, 0, 0, 1, 1},
    new int[]{2, 1, 0, 0, 1, 0, 2, 1, 2}, new int[]{2, 1, 0, 0, 1, 1, 0, 1, 0},
    new int[]{2, 1, 0, 0, 1, 2, 0, 1, 2}, new int[]{2, 1, 0, 0, 1, 2, 2, 1, 0},
    new int[]{2, 1, 0, 2, 1, 0, 0, 1, 2}, new int[]{2, 1, 0, 2, 1, 2, 0, 1, 0},
    new int[]{2, 1, 1, 0, 0, 1, 0, 0, 1}, new int[]{2, 1, 1, 0, 1, 0, 0, 1, 0},
    new int[]{2, 1, 1, 0, 1, 0, 1, 0, 0}, new int[]{2, 1, 2, 0, 1, 0, 0, 1, 2},
    new int[]{2, 1, 2, 0, 1, 0, 2, 1, 0}, new int[]{2, 1, 2, 0, 1, 2, 0, 1, 0},
    new int[]{2, 1, 2, 2, 1, 0, 0, 1, 0}, new int[]{2, 2, 0, 0, 0, 2, 1, 1, 1},
    new int[]{2, 2, 0, 0, 2, 0, 1, 1, 1}, new int[]{2, 2, 0, 1, 1, 1, 0, 0, 2},
    new int[]{2, 2, 0, 1, 1, 1, 0, 2, 0}, new int[]{2, 2, 0, 1, 1, 1, 2, 0, 0},
    new int[]{2, 2, 0, 2, 0, 0, 1, 1, 1}, new int[]{2, 2, 1, 0, 0, 1, 0, 2, 1},
    new int[]{2, 2, 1, 0, 0, 1, 2, 0, 1}, new int[]{2, 2, 1, 0, 1, 0, 1, 0, 2},
    new int[]{2, 2, 1, 0, 1, 0, 1, 2, 0}, new int[]{2, 2, 1, 0, 1, 2, 1, 0, 0},
    new int[]{2, 2, 1, 0, 2, 1, 0, 0, 1}, new int[]{2, 2, 1, 2, 0, 1, 0, 0, 1},
    new int[]{2, 2, 1, 2, 1, 0, 1, 0, 0}, new int[]{0, 0, 1, 1, 0, 0, 0, 1, 1},
    new int[]{0, 0, 1, 1, 1, 0, 0, 0, 1}, new int[]{0, 0, 1, 1, 1, 0, 0, 1, 0},
    new int[]{0, 1, 0, 0, 0, 1, 1, 0, 1}, new int[]{0, 1, 0, 0, 1, 0, 1, 0, 1},
    new int[]{0, 1, 0, 0, 1, 1, 1, 0, 0}, new int[]{0, 1, 0, 1, 0, 0, 1, 0, 1},
    new int[]{0, 1, 0, 1, 1, 0, 0, 0, 1}, new int[]{0, 1, 1, 1, 0, 0, 0, 0, 1},
    new int[]{1, 0, 0, 0, 0, 1, 1, 1, 0}, new int[]{1, 0, 0, 0, 1, 1, 0, 1, 0},
    new int[]{1, 0, 0, 0, 1, 1, 1, 0, 0}, new int[]{1, 0, 1, 0, 0, 1, 0, 1, 0},
    new int[]{1, 0, 1, 0, 1, 0, 0, 1, 0}, new int[]{1, 0, 1, 1, 0, 0, 0, 1, 0},
    new int[]{1, 1, 0, 0, 0, 1, 1, 0, 0}};

private static double[] weights = new double[]{-0.00000000000000063401,
    0.00000000000000055700, 0.00000000000000012769, -0.52573653474162341000,
    0.43427498705107342000, 0.09146154769055023200, 0.00000000000000138130,
    -0.00000000000000118053, -0.00000000000000050631, 0.52573653474162607000,
    -0.43427498705107603000, -0.09146154769055094000, -0.00000000000000057743,
    0.00000000000000037314, -0.00000000000000023441, 0.52573653474162907000,
    -0.43427498705107787000, -0.09146154769055155100, -0.00000000000000405476,
    0.00000000000000339568, 0.00000000000000053496, -0.52573653474162763000,
    0.43427498705107587000, 0.09146154769055155100, -0.00000000000000116499,
    0.00000000000000111960, 0.00000000000000004464, 0.59181480684449950000,
    -0.48617039139374285000, -0.10564441545075645000, 0.33659693927260309000,
    -0.28023189914604213000, -0.05636504012656110000, -0.00000000000000339401,
    0.00000000000000312093, 0.00000000000000057542, 0.33659693927260292000,
    -0.28023189914604213000, -0.05636504012656087800, 0.00000000000000099480,
    -0.00000000000000067295, -0.00000000000000003901, -0.33659693927260537000,
    0.28023189914604435000, 0.05636504012656118300, -0.00000000000000284785,
    0.00000000000000269180, 0.00000000000000026089, -0.33659693927260426000,
    0.28023189914604330000, 0.05636504012656121800, -0.59181480684449039000,
    0.48617039139373414000, 0.10564441545075609000, 0.00000000000000098567,
    -0.00000000000000095474, -0.00000000000000021207, -0.33659693927260698000,
    0.28023189914604579000, 0.05636504012656142600, -0.59181480684449372000,
    0.48617039139373774000, 0.10564441545075645000, 0.33659693927260514000,
    -0.28023189914604435000, -0.05636504012656100300, -0.00000000000000010012,
    0.00000000000000001702, 0.00000000000000012437, -0.33659693927260204000,
    0.28023189914604152000, 0.05636504012656010100, 0.59181480684449428000,
    -0.48617039139373813000, -0.10564441545075638000, 0.33659693927260081000,
```

```
      -0.28023189914603991000, -0.05636504012656074600, 0.00000000000000216976,
      -0.00000000000000195478, -0.00000000000000023527, 0.39961448116107012000,
      -0.35734834346184241000, -0.04226613769922773400, -0.33634249144114892000,
      0.28239332896420155000, 0.05394916247694748300, 0.39961448116106396000,
      -0.35734834346183769000, -0.04226613769922723400, -0.33634249144114703000,
      0.28239332896420027000, 0.05394916247694724100, -0.21667948075941171000,
      0.12935693076722185000, 0.08732254999219028800, -0.33634249144114398000,
      0.28239332896419722000, 0.05394916247694688700, 0.39961448116106157000,
      -0.35734834346183453000, -0.04226613769922710200, -0.33634249144114919000,
      0.28239332896420105000, 0.05394916247694810100, 0.39961448116107307000,
      -0.35734834346184485000, -0.04226613769922824700, -0.54188833749531484000,
      0.49456532031183192000, 0.04732301718348254400, 0.00000000000000042643,
      -0.00000000000000052416, -0.00000000000000028161, 0.54188833749532672000,
      -0.49456532031184147000, -0.04732301718348516700, 0.00000000000000208148,
      -0.00000000000000170526, -0.00000000000000039120, -0.00000000000001165642,
      0.00000000000000998830, 0.00000000000000133016, -0.00000000000000389738,
      0.00000000000000286692, 0.00000000000000081238, 0.54188833749532805000,
      -0.49456532031184208000, -0.04732301718348581200, -0.00000000000000308117,
      0.00000000000000212213, 0.00000000000000117840, -0.54188833749532439000,
      0.49456532031183975000, 0.04732301718348420900, 0.20000000000000001000,
      0.20000000000000001000, 0.20000000000000001000, 0.20000000000000001000,
      0.20000000000000001000, 0.20000000000000001000, 0.20000000000000001000,
      0.20000000000000001000, 0.20000000000000001000, 0.20000000000000001000,
      0.20000000000000001000, 0.20000000000000001000, 0.20000000000000001000,
      0.20000000000000001000, 0.20000000000000001000, 0.33333333333333331000,
      0.33333333333333331000, 0.33333333333333331000, 0.00000000000000093850,
      -0.00000000000000054323, -0.00000000000000011761, -0.03290466729806285100,
      0.00000000000000063771, 0.00000000000000000000, 0.00000000000000000000,
      0.00000000000000000000, 0.00000000000000000000};

   // **********************************************************************
   // MAIN
   // **********************************************************************
   [STAThread]
   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 >= 626?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;
stageITrainer.MaximumStepsize = 10.0;
stageIITrainer.MaximumStepsize = 10.0;
```

```
stageITrainer.RelativeTolerance = 10e-20;
stageIITrainer.RelativeTolerance = 10e-20;
stageIITrainer.MaximumTrainingIterations = 8000;
EpochTrainer trainer = new EpochTrainer(stageITrainer, stageIITrainer);

// Set Training Parameters
trainer.NumberOfEpochs = 20;
trainer.EpochSize = nObs;

// Set random number seeds to produce repeatable output
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(
   "**********************************************");
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 = " +
         trainer.ErrorValue);
      System.Console.Out.WriteLine("StageITrainer.getErrorValue = " +
         stageITrainer.ErrorValue);
      System.Console.Out.WriteLine("StageIITrainer.getErrorValue = " +
         stageIITrainer.ErrorValue);
   }
}
```

Output

```
trainer.getErrorValue = 2.70336729506627
StageITrainer.getErrorValue = 341.896192465939
StageIITrainer.getErrorValue = 2.70336729506627
**********************************************
--> Cross-entropy error:        2.703367
--> Classification error rate:  0.001043841
**********************************************

           Weights              Gradients
  0   -23.3898268764697    -1.68029704681242E-07
  1    55.9787793904324    -1.08507148869191E-08
  2   -24.8145508610809     9.8061928454764E-08
  3     0.614823127009218   3.50998965231112E-12
  4    17.1865318394712     2.10565444376703E-08
  5    26.0755599867853     5.06729853419313E-07
  6   -79.662169064793      6.25628057736701E-08
  7    31.9503564401075     6.44400775283152E-08
  8    -6.64353849087844   -1.10674419505296E-07
  9     4.04220907102592    3.58640760846773E-08
```

10	1.0527872483246	-3.67933513653383E-13
11	13.0870400255061	2.45188464096435E-12
12	-6.44563962176012	-8.91487913117069E-14
13	4.73560655651027	-4.35023770427128E-14
14	-24.9264785301127	6.20214116363857E-08
15	-23.6427881573418	-1.14477885305923E-07
16	36.1142420440452	6.99220259813404E-08
17	-29.9902559401382	1.4414069123191E-08
18	4.15060933344903	-1.10674419501773E-07
19	16.437715509481	6.08188216513216E-08
20	27.4813508997559	4.53177666110481E-07
21	-53.6879791476782	-1.82074832099484E-08
22	39.3419246815713	1.48087847711172E-07
23	-1.32095116065861	3.46648520684838E-12
24	-4.97262689480514	5.81232105074117E-08
25	1.14481243145803	-1.05839587128489E-21
26	7.15863775328666	1.04188938242176E-27
27	-6.50772644821445	-7.53622640287779E-20
28	-2.92357651222831	-1.45472001674375E-18
29	-16.1560422854638	2.39290821022935E-30
30	-23.4993100767818	5.79605993203358E-07
31	52.0343349139142	7.45016076734971E-08
32	-28.2488122880172	6.57361274940986E-08
33	10.290370958881	3.50959701082581E-12
34	-14.1841875743948	5.69151888883751E-08
35	26.5429154715926	-2.40906212406617E-07
36	-63.5315861748381	-2.27895167867453E-08
37	27.100860517307	9.67657893402628E-08
38	-11.4310325669034	-1.10674419114656E-07
39	4.74216035114659	5.4316339724485E-12
40	0.641561677302889	7.8151944735487E-18
41	2.22041564722028	2.45188464018683E-12
42	3.14740658020421	-7.3583066857398E-20
43	1.60525904596141	-4.35003753468099E-14
44	4.19441207505956	6.20214116363857E-08
45	-24.8603049259788	2.51683117079402E-07
46	48.5005588749859	-3.11319283726811E-08
47	-24.542863895475	1.09800516755541E-07
48	-18.3502082374005	-1.10670953396668E-07
49	29.5808529397307	9.78854869310201E-08
50	27.0178339397479	8.70166637251589E-08
51	-63.5245890293031	8.2846471144073E-08
52	31.8186019735075	5.27014000787473E-08
53	2.42002770353186	3.78644428187296E-16
54	-9.56099146066506	2.10565452277132E-08
55	1.94510648915742	-3.83094161077916E-21
56	5.22881170331311	2.00481046276046E-23
57	-6.02717816056349	-3.61963925006449E-22
58	17.0022173292331	2.60785016867033E-21
59	-24.6479339050803	5.75069913163228E-24
60	-15.5725169324771	4.63876063739422E-07
61	44.0374583067424	-4.69442836408846E-08
62	-41.1769305924502	1.48404176921322E-07
63	25.7233629013922	-1.10670909893662E-07
64	-41.2586564362978	5.43017535695804E-12
65	31.6706303703039	-1.23445417101149E-07

```
 66   -68.6519628747457     9.86588264122765E-08
 67    43.6571564988015     1.44141584994042E-08
 68   -10.5229586851025    -4.31243584579288E-14
 69    18.0830989294685     1.189366019836E-07
 70   -12.8256099315956    -1.73086583371581E-09
 71    14.2751176420669     2.82904280520231E-26
 72     0.611612996756902  -3.16418586438729E-10
 73   -14.5866509581701     9.97653170130876E-23
 74    18.3504579166293    -2.2387178617835E-19
 75   -23.3888491998405    -1.23539736925098E-07
 76    52.2277746589779     1.07001235079262E-07
 77   -26.1442463831178     1.52827861992725E-08
 78   -12.0817712463414     3.78108725218232E-16
 79   -16.732006840824      6.08145629733916E-08
 80    27.2862196328307     4.6223951772184E-07
 81   -60.2711282018116    -5.52891441925098E-08
 82    30.6956745853904     1.47219130635017E-07
 83     3.66731027306885   -1.106709098943E-07
 84     3.53129758498572   -3.89394245104408E-09
 85     0.600371582483001   7.81479027111874E-18
 86    -1.8657171315698     2.4518846401873E-12
 87    -2.60420501014975   -2.04744229740536E-21
 88     8.27516103748487   -4.3501828950192E-14
 89     8.23539794931856    6.20214116363857E-08
 90   -23.1541029813632     5.31093839398311E-08
 91    70.6668233987974     1.44581007663968E-07
 92   -17.539510220755      5.15328591142091E-08
 93     1.95410802312735   -1.10674462625394E-07
 94     3.06783205444343    5.81286424613412E-08
 95    27.7056768559965     2.87321262698441E-07
 96   -59.9447333420135    -9.28664648925763E-08
 97    35.1178032203679     1.11285476332644E-07
 98   -18.7545966339525     3.50960791885221E-12
 99    -3.09065236278368    6.0813389697616E-08
100    -1.35361646970208   -1.73086583371581E-09
101   -20.63501003417       1.47945976887397E-34
102   -15.2390999588768    -3.16418612564994E-10
103    17.2768473527366    -5.45484469543474E-19
104    -4.20098762033162   -2.23866242943624E-19
105   -24.9467968472952     2.47563888504666E-07
106    32.4810122215473    -1.82099350933995E-08
107   -20.9181167195218     1.48087751691707E-07
108    -5.34209226786082    3.50998612937016E-12
109    -9.4479410160346     1.71571720581279E-08
110    27.3556997100585     9.11358922998901E-08
111   -52.6175860338615     6.99244778647915E-08
112    24.5933226669344     1.44141651425814E-08
113    -4.74888992948916   -1.10674463004154E-07
114    -4.74447523527827    1.01784860100605E-07
115     1.3068899966691     3.38131779070714E-24
116     9.84279576559971    2.93298503589978E-26
117    -2.65752552540913   -7.47305201166442E-23
118     9.77370532097723    3.71315199958786E-21
119    10.1683911508445     1.55853874772282E-25
120   -22.9734431714424     5.07169516120168E-07
121    63.2201329935377    -3.03012043165043E-08
```

122	-25.9684891770425	1.61822205642615E-07
123	1.72757810305429	-1.10674462618761E-07
124	-22.9539941930863	9.7885487944738E-08
125	26.0467643486586	-1.6673923742323E-07
126	-70.3904826524733	8.20157470878963E-08
127	38.3501514185098	9.96040681647748E-10
128	-7.89538750972399	3.50960128669369E-12
129	8.61060695952724	2.10565442142192E-08
130	0.334837626579686	-1.73049789238167E-09
131	-3.00049387697595	3.09824433715878E-24
132	-11.83638121782	-3.16329489975111E-10
133	6.99299819439263	-5.4590838025404E-19
134	9.24272160689922	-2.23860613161707E-19
135	-130.359444244494	-1.82459707733754E-08
136	-28.4061185197319	-2.86376914471931E-09
137	-5.68499307904826	2.43550052478779E-08
138	16.0406331984156	-1.80938585140171E-09
139	64.167895394211	1.4852818776177E-09
140	-60.12501945685	-2.9636372270146E-09
141	-10.1852707596995	1.1409812574327E-08
142	-82.7498540101702	9.97082548032286E-09
143	105.503273325536	-2.44596919012615E-09
144	36.6129197770311	-2.28569807296338E-08
145	38.4606669887555	-5.22871278391192E-09
146	151.846032339171	-7.48974152293252E-10
147	-66.0526100545884	3.48565829108654E-08
148	30.1143870019644	-5.6524446126533E-09
149	-74.5243364539271	-1.91528201142895E-09
150	142.594614534703	-7.05098701787417E-08
151	93.6715421122173	1.00751410014421E-09
152	-104.702722956492	-6.79330110522797E-08
153	3.81041225574935	3.38699780804557E-07
154	-9.54448888867617	5.17145427713919E-08
155	1.82018284630767	1.62501916834288E-07
156	0.359750818534005	-1.10670953018021E-07
157	-5.30067023962807	1.18942032158733E-07
158	78.9569244147036	7.53075424823755E-09
159	57.9483835349265	9.5192378081288E-09
160	3.61286655852224	3.24256332562239E-08
161	-36.2824537467563	-6.92558401759917E-08

	Forecast	
	Expected	Predicted
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1
5	1	1
6	1	1
7	1	1
8	1	1
9	1	1
10	1	1
11	1	1
12	1	1

```
13      1          1
14      1          1
15      1          1
16      1          1
17      1          1
18      1          1
19      1          1
20      1          1
21      1          1
22      1          1
23      1          1
24      1          1
25      1          1
26      1          1
27      1          1
28      1          1
29      1          1
30      1          1
31      1          1
32      1          1
33      1          1
34      1          1
35      1          1
36      1          1
37      1          1
38      1          1
39      1          1
40      1          1
41      1          1
42      1          1
43      1          1
44      1          1
45      1          1
46      1          1
47      1          1
48      1          1
49      1          1
50      0          0
51      0          0
52      0          0
53      0          0
54      0          0
55      0          0
56      0          0
57      0          0
58      0          0
59      0          0
60      0          0
61      0          0
62      0          0
63      0          0
64      0          0
65      0          0
66      0          0
67      0          0
68      0          0
```

69	0	0
70	0	0
71	0	0
72	0	0
73	0	0
74	0	0
75	0	0
76	0	0
77	0	0
78	0	0
79	0	0
80	0	0
81	0	0
82	0	0
83	0	0
84	0	0
85	0	0
86	0	0
87	0	0
88	0	0
89	0	0
90	0	0
91	0	0
92	0	0
93	0	0
94	0	0
95	0	0
96	0	0
97	0	0
98	0	0
99	0	0
100	0	0
101	0	0
102	0	0
103	0	0
104	0	0
105	0	0
106	0	0
107	0	0
108	0	0
109	0	0
110	0	0
111	0	0
112	0	0
113	0	0
114	0	0
115	0	0
116	0	0
117	0	0
118	0	0
119	0	0
120	0	0
121	0	0
122	0	0
123	0	0
124	0	0

125	0	0
126	0	0
127	0	0
128	0	0
129	0	0
130	0	0
131	0	0
132	0	0
133	0	0
134	0	0
135	0	0
136	0	0
137	0	0
138	0	0
139	0	0
140	0	0
141	0	0
142	0	0
143	0	0
144	0	0
145	0	0
146	0	0
147	0	0
148	0	0
149	0	0
150	0	0
151	0	0
152	0	0
153	0	0
154	0	0
155	0	0
156	0	0
157	0	0
158	0	0
159	0	0
160	0	0
161	0	0
162	0	0
163	0	0
164	0	0
165	0	0
166	0	0
167	0	0
168	0	0
169	0	0
170	0	0
171	0	0
172	0	0
173	0	0
174	0	0
175	0	0
176	0	0
177	0	0
178	0	0
179	0	0
180	0	0

181	0	0
182	0	0
183	0	0
184	0	0
185	0	0
186	0	0
187	0	0
188	0	0
189	0	0
190	0	0
191	0	0
192	0	0
193	0	0
194	0	0
195	0	0
196	0	0
197	0	0
198	0	0
199	0	0
200	0	0
201	0	0
202	0	0
203	0	0
204	0	0
205	0	0
206	0	0
207	0	0
208	0	0
209	0	0
210	0	0
211	0	0
212	0	0
213	0	0
214	0	0
215	0	0
216	0	0
217	0	0
218	0	0
219	0	0
220	0	0
221	0	0
222	0	0
223	0	0
224	0	0
225	0	0
226	0	0
227	0	0
228	0	0
229	0	0
230	0	0
231	0	0
232	0	0
233	0	0
234	0	0
235	0	0
236	0	0

237	0	0
238	0	0
239	0	0
240	0	0
241	0	0
242	0	0
243	0	0
244	0	0
245	0	0
246	0	0
247	0	0
248	0	0
249	0	0
250	0	0
251	0	0
252	0	0
253	0	0
254	0	0
255	0	0
256	0	0
257	0	0
258	0	0
259	0	0
260	0	0
261	0	0
262	0	0
263	0	0
264	0	0
265	0	0
266	0	0
267	0	0
268	0	0
269	0	0
270	0	0
271	0	0
272	0	0
273	0	0
274	0	0
275	0	0
276	0	0
277	0	0
278	0	0
279	0	0
280	0	0
281	0	0
282	0	0
283	0	0
284	0	0
285	0	0
286	0	0
287	0	0
288	0	0
289	0	0
290	0	0
291	0	0
292	0	0

293	0	0
294	0	0
295	0	0
296	0	0
297	0	0
298	0	0
299	0	0
300	0	0
301	0	0
302	0	0
303	0	0
304	0	0
305	0	0
306	0	0
307	0	0
308	0	0
309	0	0
310	0	0
311	0	0
312	0	0
313	0	0
314	0	0
315	0	0
316	0	0
317	0	0
318	0	0
319	0	0
320	0	0
321	0	0
322	0	0
323	0	0
324	0	0
325	0	0
326	0	0
327	0	0
328	0	0
329	0	0
330	0	0
331	0	0
332	0	0
333	0	0
334	0	0
335	0	0
336	0	0
337	0	0
338	0	0
339	0	0
340	0	0
341	0	0
342	0	0
343	0	0
344	0	0
345	0	0
346	0	0
347	0	0
348	0	0

349	0	0
350	0	0
351	0	0
352	0	0
353	0	0
354	0	0
355	0	0
356	0	0
357	0	0
358	0	0
359	0	0
360	0	0
361	0	0
362	0	0
363	0	0
364	0	0
365	0	0
366	0	0
367	0	0
368	0	0
369	0	0
370	0	0
371	0	0
372	0	0
373	0	0
374	0	0
375	0	0
376	0	0
377	0	0
378	0	0
379	0	0
380	0	0
381	0	0
382	0	0
383	0	0
384	0	0
385	0	0
386	0	0
387	0	0
388	0	0
389	0	0
390	0	0
391	0	0
392	0	0
393	0	0
394	0	0
395	0	0
396	0	0
397	0	0
398	0	0
399	0	0
400	0	0
401	0	0
402	0	0
403	0	0
404	0	0

405	0	0
406	0	0
407	0	0
408	0	0
409	0	0
410	0	0
411	0	0
412	0	0
413	0	0
414	0	0
415	0	0
416	0	0
417	0	0
418	0	0
419	0	0
420	0	0
421	0	0
422	0	0
423	0	0
424	0	0
425	0	0
426	0	0
427	0	0
428	0	0
429	0	0
430	0	0
431	0	0
432	0	0
433	0	0
434	0	0
435	0	0
436	0	0
437	0	0
438	0	0
439	0	0
440	0	0
441	0	0
442	0	0
443	0	0
444	0	0
445	0	0
446	0	0
447	0	0
448	0	0
449	0	0
450	0	0
451	0	0
452	0	0
453	0	0
454	0	0
455	0	0
456	0	0
457	0	0
458	0	0
459	0	0
460	0	0

461	0	0
462	0	0
463	0	0
464	0	0
465	0	0
466	0	0
467	0	0
468	0	0
469	0	0
470	0	0
471	0	0
472	0	0
473	0	0
474	0	0
475	0	0
476	0	0
477	0	0
478	0	0
479	0	0
480	0	0
481	0	0
482	0	0
483	0	0
484	0	0
485	0	0
486	0	0
487	0	0
488	0	0
489	0	0
490	0	0
491	0	0
492	0	0
493	0	0
494	0	0
495	0	0
496	0	0
497	0	0
498	0	0
499	0	0
500	0	0
501	0	0
502	0	0
503	0	0
504	0	0
505	0	0
506	0	0
507	0	0
508	0	0
509	0	0
510	0	0
511	0	0
512	0	0
513	0	0
514	0	0
515	0	0
516	0	0

517	0	0
518	0	0
519	0	0
520	0	0
521	0	0
522	0	0
523	0	0
524	0	0
525	0	0
526	0	0
527	0	0
528	0	0
529	0	0
530	0	0
531	0	0
532	0	0
533	0	0
534	0	0
535	0	0
536	0	0
537	0	0
538	0	0
539	0	0
540	0	0
541	0	0
542	0	0
543	0	0
544	0	0
545	0	0
546	0	0
547	0	0
548	0	0
549	0	0
550	0	0
551	0	0
552	0	0
553	0	0
554	0	0
555	0	0
556	0	0
557	0	0
558	0	0
559	0	0
560	0	0
561	0	0
562	0	0
563	0	0
564	0	0
565	0	0
566	0	0
567	0	0
568	0	0
569	0	0
570	0	0
571	0	0
572	0	0

573	0	0
574	0	0
575	0	0
576	0	0
577	0	0
578	0	0
579	0	0
580	0	0
581	0	0
582	0	0
583	0	0
584	0	0
585	0	0
586	0	0
587	0	0
588	0	0
589	0	0
590	0	0
591	0	0
592	0	0
593	0	0
594	0	0
595	0	0
596	0	0
597	0	0
598	0	0
599	0	0
600	0	0
601	0	0
602	0	0
603	0	0
604	0	0
605	0	0
606	0	0
607	0	0
608	0	0
609	0	0
610	0	0
611	0	0
612	0	0
613	0	0
614	0	0
615	0	0
616	0	0
617	0	0
618	0	0
619	0	0
620	0	0
621	0	0
622	0	0
623	0	0
624	0	0
625	0	0
626	0	0
627	0	0
628	0	0

629	0	0
630	0	0
631	0	0
632	0	0
633	0	0
634	0	0
635	0	0
636	0	0
637	0	0
638	0	0
639	0	0
640	0	0
641	0	0
642	0	0
643	0	0
644	0	0
645	0	0
646	0	0
647	0	0
648	0	0
649	0	0
650	0	0
651	0	0
652	0	0
653	0	0
654	0	0
655	0	0
656	0	0
657	0	0
658	0	0
659	0	0
660	0	0
661	0	0
662	0	0
663	0	0
664	0	0
665	0	0
666	0	0
667	0	0
668	0	0
669	0	0
670	0	0
671	0	0
672	0	0
673	0	0
674	0	0
675	0	0
676	0	0
677	0	0
678	0	0
679	0	0
680	0	0
681	0	0
682	0	0
683	0	0
684	0	0

685	0	0
686	0	0
687	0	0
688	0	0
689	0	0
690	0	0
691	0	0
692	0	0
693	0	0
694	0	0
695	0	0
696	0	0
697	0	0
698	0	0
699	0	0
700	0	0
701	0	0
702	0	0
703	0	0
704	0	0
705	0	0
706	0	0
707	0	0
708	0	0
709	0	0
710	0	0
711	0	0
712	0	0
713	0	0
714	0	0
715	0	0
716	0	0
717	0	0
718	0	0
719	0	0
720	0	0
721	0	0
722	0	0
723	0	0
724	0	0
725	0	0
726	0	0
727	0	0
728	0	0
729	0	0
730	0	0
731	0	0
732	0	0
733	0	0
734	0	0
735	0	0
736	0	0
737	0	0
738	0	0
739	0	0
740	0	0

741	0	0
742	0	0
743	0	0
744	0	0
745	0	0
746	0	0
747	0	0
748	0	0
749	0	0
750	0	0
751	0	0
752	0	0
753	0	0
754	0	0
755	0	0
756	0	0
757	0	0
758	0	0
759	0	0
760	0	0
761	0	0
762	0	0
763	0	0
764	0	0
765	0	0
766	0	0
767	0	0
768	0	0
769	0	0
770	0	0
771	0	0
772	0	0
773	0	0
774	0	0
775	0	0
776	0	0
777	0	0
778	0	0
779	0	0
780	0	0
781	0	0
782	0	0
783	0	0
784	0	0
785	0	0
786	0	0
787	0	0
788	0	0
789	0	0
790	0	0
791	0	0
792	0	0
793	0	0
794	0	0
795	0	0
796	0	0

797	0	0
798	0	0
799	0	0
800	0	0
801	0	0
802	0	0
803	0	0
804	0	0
805	0	0
806	0	0
807	0	0
808	0	0
809	0	0
810	0	0
811	0	0
812	0	0
813	0	0
814	0	0
815	0	0
816	0	0
817	0	0
818	0	0
819	0	0
820	0	0
821	0	0
822	0	0
823	0	0
824	0	0
825	0	0
826	0	0
827	0	0
828	0	0
829	0	0
830	0	0
831	0	0
832	0	0
833	0	0
834	0	0
835	0	0
836	0	0
837	0	0
838	0	0
839	0	0
840	0	0
841	0	0
842	0	0
843	0	0
844	0	0
845	0	0
846	0	0
847	0	0
848	0	0
849	0	0
850	0	0
851	0	0
852	0	0

853	0	0
854	0	0
855	0	0
856	0	0
857	0	0
858	0	0
859	0	0
860	0	0
861	0	0
862	0	0
863	0	0
864	0	0
865	0	0
866	0	0
867	0	0
868	0	0
869	0	0
870	0	0
871	0	0
872	0	0
873	0	0
874	0	0
875	0	0
876	0	0
877	0	0
878	0	0
879	0	0
880	0	0
881	0	0
882	0	0
883	0	0
884	0	0
885	0	0
886	0	0
887	0	0
888	0	0
889	0	0
890	0	0
891	0	0
892	0	0
893	0	0
894	0	0
895	0	0
896	0	0
897	0	0
898	0	0
899	0	0
900	0	0
901	0	0
902	0	0
903	0	0
904	0	0
905	0	0
906	0	0
907	0	0
908	0	0

```
909     0           0
910     0           0
911     0           0
912     0           0
913     0           0
914     0           0
915     0           0
916     0           0
917     0           0
918     0           0
919     0           0
920     0           0
921     0           0
922     0           0
923     0           0
924     0           0
925     0           0
926     0           0
927     0           0
928     0           0
929     0           0
930     0           0
931     0           0
932     0           0
933     0           0
934     0           0
935     0           0
936     0           0
937     0           0
938     0           0
939     0           0
940     0           0
941     0           0
942     0           0
943     0           0
944     0           0
945     0           0
946     0           0
947     0           0
948     0           0
949     0           0
950     0           0
951     0           0
952     0           0
953     0           0
954     0           0
955     0           0
956     0           0
957     0           0

****************Time:  22.859
trainer.getErrorValue = 2.70336729506627
StageITrainer.getErrorValue = 341.896192465939
StageIITrainer.getErrorValue = 2.70336729506627
```

MultiClassification Class

Summary

Classifies patterns into three or more classes.

`public class Imsl.DataMining.Neural.MultiClassification`

Properties

Error

`virtual public Imsl.DataMining.Neural.QuasiNewtonTrainer.IError Error {get; }`

Description

The error function for use by `QuasiNewtonTrainer` for training a classification network.

This error function combines the softmax activation function and the cross-entropy error function.

Network

`virtual public Imsl.DataMining.Neural.Network Network {get; }`

Description

Returns the network being used for classification.

Constructor

MultiClassification

`public MultiClassification(Imsl.DataMining.Neural.Network network)`

Description

Creates a classifier.

Parameter

`network` – Is the neural network used for classification. Its output perceptrons should use linear activation functions. The number of output perceptrons should equal the number of classes.

Methods

ComputeStatistics

`virtual public double[] ComputeStatistics(double[,] xData, int[] yData)`

Description

Computes classification statistics for the supplied network patterns and their associated classifications.

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.

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 number of columns in *yData* must equal the number of `Perceptrons` in the `OutputLayer`.

Returns

A `double[]` containing the cross-entropy error and the classification error rate.

PredictedClass

```
virtual public int PredictedClass(double[] x)
```

Description

Calculates the classification probablities for the input pattern x, and returns the class with the highest probability.

This method classifies patterns into one of the target classes based upon the patterns values.

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.

Probabilities

```
virtual public double[] Probabilities(double[] x)
```

Description

Returns classification probabilities for the input pattern x.

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 output perceptrons. The softmax function transforms the outputs potential z to the probability y by

$$y_i = \mathrm{softmax_i} = \frac{\mathrm{e}^{Z_i}}{\sum_{j=1}^{C} e^{Z_j}}$$

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.

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 Imsl.DataMining.Neural.MultiClassification.Error (p. 1089) 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 values. These values must be in the range of one to the number of output perceptrons in the network.

Description

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.

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.

```
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[]{4.8, 3.0, 1.4, 0.1, 1}, new double[]{4.3, 3.0, 1.1, 0.1, 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[]{5.7, 3.8, 1.7, 0.3, 1}, new double[]{5.1, 3.8, 1.5, 0.3, 1},
      new double[]{5.4, 3.4, 1.7, 0.2, 1}, new double[]{5.1, 3.7, 1.5, 0.4, 1},
      new double[]{4.6, 3.6, 1.0, 0.2, 1}, new double[]{5.1, 3.3, 1.7, 0.5, 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, 3.4, 1.4, 0.2, 1}, new double[]{4.7, 3.2, 1.6, 0.2, 1},
      new double[]{4.8, 3.1, 1.6, 0.2, 1}, new double[]{5.4, 3.4, 1.5, 0.4, 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},
   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}};

[STAThread]
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(
      "**********************************************");
   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("");

   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();
      pmf.SetColumnLabels( new System.String[]{"Sepal Length", "Sepal Width",
         "Petal Length", "Petal Width", "Expected", "Predicted"});
      new PrintMatrix("Forecast").Print(pmf, report);


      // **********************************************************************
      // DISPLAY CLASSIFICATION STATISTICS
      // **********************************************************************
      double[] statsClass = classification.ComputeStatistics(xData, yData);
      // Display Network Errors
      System.Console.Out.WriteLine(
         "**********************************************");
      System.Console.Out.WriteLine("--> Cross-Entropy Error:      " +
         (float)statsClass[0]);
      System.Console.Out.WriteLine("--> Classification Error:     " +
         (float)statsClass[1]);
      System.Console.Out.WriteLine(
         "**********************************************");
      System.Console.Out.WriteLine("");
      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("Cross-Entropy Error Value = " +
         trainer.ErrorValue);
   }
}
```

Output

```
**********************************************
--> Cross-entropy error:        2.119975E-10
--> Classification error rate:  0
**********************************************

           Weights              Gradients
```

0	-28.6781319150959	7.87590264437711E-113
1	-1.04608488867643	-5.76760919985582E-12
2	-86.2597201959254	-6.79823502891189E-181
3	-80.8768794497344	1.91947652160099E-08
4	4.7922023557326	5.51313185106398E-113
5	41.7757062372833	-2.60180626610986E-12
6	-89.1920834776831	-3.52964111087987E-181
7	-169.665870083246	1.04107297570942E-08
8	26.2104571153094	2.52028884620068E-113
9	89.2824428920942	-4.59704885126527E-12
10	4.78968378602374	-5.09809934877317E-181
11	3.69580856206968	1.56160846865524E-08
12	199.180907855982	9.45108317325254E-114
13	-81.462945734365	-1.43284780406862E-12
14	234.395038164347	-1.82996826335742E-181
15	523.701133894724	5.85603331200232E-09
16	3.75750825503756	-7.63631139063138E-11
17	1.55131869241627	-8.20696721896107E-08
18	-0.478073600231926	-1.83171284540763E-09
19	82.4910685116272	-2.58054098695318E-16
20	2.49879733323649	-3.60921285939333E-13
21	-0.114220454165923	-7.89173652189626E-15
22	-29.1499708696859	-9.79535645888544E-186
23	12.7239135762807	-1.0637578300168E-182
24	-18.4461484012531	-2.33659980816857E-184
25	-8.04900610619496	4.22051830796542E-14
26	41.9059823374628	8.05687997104916E-11
27	6.21226036008435	6.32025904169135E-13
28	-2095.35972014252	2.05965389206269E-10
29	557.990634349718	-2.11962286858389E-10
30	1538.36908579337	5.99730606961734E-12
31	-2034.36189491703	4.73007517080961E-11
32	276.908671633113	-5.08319254134931E-11
33	1758.45322328515	3.53127914432977E-12
34	-2095.02402775956	2.05287816017904E-10
35	529.488332076977	-2.11269543435035E-10
36	1566.53569568216	5.9821345130617E-12
37	-35.2271614745031	1.57518052887542E-113
38	-437.337384643647	-9.19418243757285E-13
39	-56.3739506402764	-1.3075596591524E-181
40	56.4556518262469	3.25335045818341E-09
41	5.10317144510891	-7.63631139063138E-11
42	-2.76178014091664	-8.20696721896107E-08
43	6.14951997983717	-1.83171284540763E-09
44	4819.53497802506	2.05994598070357E-10
45	-953.548516064939	-2.11992645660075E-10
46	-3865.98646195991	5.99845606447974E-12

Forecast

	Sepal Length	Sepal Width	Petal Length	Petal Width	Expected	Predicted
0	5.1	3.5	1.4	0.2	1	1
1	4.9	3	1.4	0.2	1	1
2	4.7	3.2	1.3	0.2	1	1
3	4.6	3.1	1.5	0.2	1	1
4	5	3.6	1.4	0.2	1	1
5	5.4	3.9	1.7	0.4	1	1

6	4.6	3.4	1.4	0.3	1	1
7	5	3.4	1.5	0.2	1	1
8	4.4	2.9	1.4	0.2	1	1
9	4.9	3.1	1.5	0.1	1	1
10	5.4	3.7	1.5	0.2	1	1
11	4.8	3.4	1.6	0.2	1	1
12	4.8	3	1.4	0.1	1	1
13	4.3	3	1.1	0.1	1	1
14	5.8	4	1.2	0.2	1	1
15	5.7	4.4	1.5	0.4	1	1
16	5.4	3.9	1.3	0.4	1	1
17	5.1	3.5	1.4	0.3	1	1
18	5.7	3.8	1.7	0.3	1	1
19	5.1	3.8	1.5	0.3	1	1
20	5.4	3.4	1.7	0.2	1	1
21	5.1	3.7	1.5	0.4	1	1
22	4.6	3.6	1	0.2	1	1
23	5.1	3.3	1.7	0.5	1	1
24	4.8	3.4	1.9	0.2	1	1
25	5	3	1.6	0.2	1	1
26	5	3.4	1.6	0.4	1	1
27	5.2	3.5	1.5	0.2	1	1
28	5.2	3.4	1.4	0.2	1	1
29	4.7	3.2	1.6	0.2	1	1
30	4.8	3.1	1.6	0.2	1	1
31	5.4	3.4	1.5	0.4	1	1
32	5.2	4.1	1.5	0.1	1	1
33	5.5	4.2	1.4	0.2	1	1
34	4.9	3.1	1.5	0.1	1	1
35	5	3.2	1.2	0.2	1	1
36	5.5	3.5	1.3	0.2	1	1
37	4.9	3.1	1.5	0.1	1	1
38	4.4	3	1.3	0.2	1	1
39	5.1	3.4	1.5	0.2	1	1
40	5	3.5	1.3	0.3	1	1
41	4.5	2.3	1.3	0.3	1	1
42	4.4	3.2	1.3	0.2	1	1
43	5	3.5	1.6	0.6	1	1
44	5.1	3.8	1.9	0.4	1	1
45	4.8	3	1.4	0.3	1	1
46	5.1	3.8	1.6	0.2	1	1
47	4.6	3.2	1.4	0.2	1	1
48	5.3	3.7	1.5	0.2	1	1
49	5	3.3	1.4	0.2	1	1
50	7	3.2	4.7	1.4	2	2
51	6.4	3.2	4.5	1.5	2	2
52	6.9	3.1	4.9	1.5	2	2
53	5.5	2.3	4	1.3	2	2
54	6.5	2.8	4.6	1.5	2	2
55	5.7	2.8	4.5	1.3	2	2
56	6.3	3.3	4.7	1.6	2	2
57	4.9	2.4	3.3	1	2	2
58	6.6	2.9	4.6	1.3	2	2
59	5.2	2.7	3.9	1.4	2	2
60	5	2	3.5	1	2	2
61	5.9	3	4.2	1.5	2	2

62	6	2.2	4	1	2	2
63	6.1	2.9	4.7	1.4	2	2
64	5.6	2.9	3.6	1.3	2	2
65	6.7	3.1	4.4	1.4	2	2
66	5.6	3	4.5	1.5	2	2
67	5.8	2.7	4.1	1	2	2
68	6.2	2.2	4.5	1.5	2	2
69	5.6	2.5	3.9	1.1	2	2
70	5.9	3.2	4.8	1.8	2	2
71	6.1	2.8	4	1.3	2	2
72	6.3	2.5	4.9	1.5	2	2
73	6.1	2.8	4.7	1.2	2	2
74	6.4	2.9	4.3	1.3	2	2
75	6.6	3	4.4	1.4	2	2
76	6.8	2.8	4.8	1.4	2	2
77	6.7	3	5	1.7	2	2
78	6	2.9	4.5	1.5	2	2
79	5.7	2.6	3.5	1	2	2
80	5.5	2.4	3.8	1.1	2	2
81	5.5	2.4	3.7	1	2	2
82	5.8	2.7	3.9	1.2	2	2
83	6	2.7	5.1	1.6	2	2
84	5.4	3	4.5	1.5	2	2
85	6	3.4	4.5	1.6	2	2
86	6.7	3.1	4.7	1.5	2	2
87	6.3	2.3	4.4	1.3	2	2
88	5.6	3	4.1	1.3	2	2
89	5.5	2.5	4	1.3	2	2
90	5.5	2.6	4.4	1.2	2	2
91	6.1	3	4.6	1.4	2	2
92	5.8	2.6	4	1.2	2	2
93	5	2.3	3.3	1	2	2
94	5.6	2.7	4.2	1.3	2	2
95	5.7	3	4.2	1.2	2	2
96	5.7	2.9	4.2	1.3	2	2
97	6.2	2.9	4.3	1.3	2	2
98	5.1	2.5	3	1.1	2	2
99	5.7	2.8	4.1	1.3	2	2
100	6.3	3.3	6	2.5	3	3
101	5.8	2.7	5.1	1.9	3	3
102	7.1	3	5.9	2.1	3	3
103	6.3	2.9	5.6	1.8	3	3
104	6.5	3	5.8	2.2	3	3
105	7.6	3	6.6	2.1	3	3
106	4.9	2.5	4.5	1.7	3	3
107	7.3	2.9	6.3	1.8	3	3
108	6.7	2.5	5.8	1.8	3	3
109	7.2	3.6	6.1	2.5	3	3
110	6.5	3.2	5.1	2	3	3
111	6.4	2.7	5.3	1.9	3	3
112	6.8	3	5.5	2.1	3	3
113	5.7	2.5	5	2	3	3
114	5.8	2.8	5.1	2.4	3	3
115	6.4	3.2	5.3	2.3	3	3
116	6.5	3	5.5	1.8	3	3
117	7.7	3.8	6.7	2.2	3	3

```
118     7.7           2.6          6.9           2.3         3          3
119     6             2.2          5             1.5         3          3
120     6.9           3.2          5.7           2.3         3          3
121     5.6           2.8          4.9           2           3          3
122     7.7           2.8          6.7           2           3          3
123     6.3           2.7          4.9           1.8         3          3
124     6.7           3.3          5.7           2.1         3          3
125     7.2           3.2          6             1.8         3          3
126     6.2           2.8          4.8           1.8         3          3
127     6.1           3            4.9           1.8         3          3
128     6.4           2.8          5.6           2.1         3          3
129     7.2           3            5.8           1.6         3          3
130     7.4           2.8          6.1           1.9         3          3
131     7.9           3.8          6.4           2           3          3
132     6.4           2.8          5.6           2.2         3          3
133     6.3           2.8          5.1           1.5         3          3
134     6.1           2.6          5.6           1.4         3          3
135     7.7           3            6.1           2.3         3          3
136     6.3           3.4          5.6           2.4         3          3
137     6.4           3.1          5.5           1.8         3          3
138     6             3            4.8           1.8         3          3
139     6.9           3.1          5.4           2.1         3          3
140     6.7           3.1          5.6           2.4         3          3
141     6.9           3.1          5.1           2.3         3          3
142     5.8           2.7          5.1           1.9         3          3
143     6.8           3.2          5.9           2.3         3          3
144     6.7           3.3          5.7           2.5         3          3
145     6.7           3            5.2           2.3         3          3
146     6.3           2.5          5             1.9         3          3
147     6.5           3            5.2           2           3          3
148     6.2           3.4          5.4           2.3         3          3
149     5.9           3            5.1           1.8         3          3

***********************************************
--> Cross-Entropy Error:      2.119975E-10
--> Classification Error:     0
***********************************************

****************Time:  0.391
Cross-Entropy Error Value = 6.3599259192415E-10
```

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 if X1 = 1

Y = 1 if X2 = 1

Y = 2 if X3 = 1

```
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[]{1.29099444873580580000,
      -0.64549722436790280000, -0.64549722436790291000, 0.00000000000000000000,
      1.11803398874989490000, -1.11803398874989470000, 0.57735026918962584000,
      0.57735026918962584000, 0.57735026918962584000, 0.33333333333333331000,
      0.33333333333333331000, 0.33333333333333331000, 0.33333333333333331000,
      0.33333333333333331000, 0.33333333333333331000, 0.33333333333333331000,
      0.33333333333333331000, 0.33333333333333331000, -0.00000000000000005851,
      -0.00000000000000005851, -0.57735026918962573000, 0.00000000000000000000,
      0.00000000000000000000, 0.00000000000000000000};

   [STAThread]
   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.FalseConvergenceTolerance = 1.0e-20;
      trainer.GradientTolerance = 1.0e-20;
      trainer.RelativeTolerance = 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
      System.Console.Out.WriteLine(
         "***********************************************");
      System.Console.Out.WriteLine("--> Cross-Entropy Error:      " +
         (float)statsClass[0]);
      System.Console.Out.WriteLine("--> Classification Error:     " +
         (float)statsClass[1]);
      System.Console.Out.WriteLine(
         "***********************************************");
      System.Console.Out.WriteLine("");
   }
}
```

Output

```
***********************************************
--> Cross-Entropy Error:      0
--> Classification Error:     0
***********************************************

         Weights               Gradients
 0   3.22142231426227   -4.5293591783678E-21
 1  -3.5155105345287     4.10418218034394E-20
 2  -1.32663590270865    1.82727922421337E-19
 3  -1.28370297625286    1.79195753880894E-14
 4   2.64877322140103   -1.36319533545496E-14
 5  -2.83341597107777    9.79393364839217E-14
 6   0.758665437713986   2.63470704168269E-40
 7   2.23842447927351   -3.41751464346434E-40
 8   4.92756335370081   -1.62387424991113E-40
 9   4.48657597190413   -1.0629493799265E-17
10  -3.22422759973224    3.06415333213106E-15
11  -0.25310873632921   -3.03621111804672E-15
12  -5.81974214297561    6.01464280711637E-17
13   4.52625109695299   -1.7338349452045E-14
14   2.29349104602264    1.71805857805814E-14
15  -2.76184230566633   -3.10717763898766E-17
16  -6.22757265046598    8.95702927705116E-15
17  10.2111662163555    -8.87555824139815E-15
18   0.77260224274218    1.79195708587302E-14
19   0.243263227911821  -1.36319123127278E-14
20   1.33938361128243    9.79395192118442E-14
21   0.261471250833615   2.07972579617572E-17
22   0.326428754937653  -5.99520433297567E-15
23  -0.587900005771228   5.94069140932869E-15
```

```
                                           Forecast
   X1  X2  X3  Y          P(C1)                 P(C2)                 P(C3)          Predicted
0  1   0   0   1  1                     8.53314291154036E-29  8.9474561285406E-21       1
1  1   0   0   1  1                     8.53314291154036E-29  8.9474561285406E-21       1
2  0   1   0   2  1.03986289808786E-17  0.999999999999997     2.97033675720822E-15      2
3  0   1   0   2  1.03986289808786E-17  0.999999999999997     2.97033675720822E-15      2
4  0   0   1   3  2.93708802897469E-41  1.2199417162444E-44   1                         3
5  0   0   1   3  2.93708802897469E-41  1.2199417162444E-44   1                         3

Cross-Entropy Error Value = 0
***********************************************
--> Cross-Entropy Error:      0
--> Classification Error:     0
***********************************************
```

ScaleFilter Class

Summary

Scales or unscales continuous data prior to its use in neural network training, testing, or forecasting.

`public class Imsl.DataMining.Neural.ScaleFilter`

Properties

Center

`virtual public double Center {get; set; }`

Description

The measure of center to be used during z-score scaling.

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.

If this property is not set then the measure of spread is computed from the data.

Constructor

ScaleFilter

`public ScaleFilter(Imsl.DataMining.Neural.ScaleFilter.ScalingMethod scalingMethod)`

Description

Constructor for `ScaleFilter`.

scalingMethod is specified by: ScalingMethod.None (p. 1114), ScalingMethod.Bounded (p. 1113), ScalingMethod.UnboundedZScoreMeanStdev (p. 1114), ScalingMethod.UnboundedZScoreMedianMAD (p. 1114), ScalingMethod.BoundedZScoreMeanStdev (p. 1113), or ScalingMethod.BoundedZScoreMedianMAD (p. 1113).

Parameter

`scalingMethod` – An `int` specifying the scaling method to be applied.

Methods

Decode

`virtual public void Decode(int columnIndex, double[,] z)`

Description

Unscales a single column of a two dimensional array of values.

Indexing is zero-based.

Its *columnIndex*-th column is modified in place.

Parameters

`columnIndex` – An `int` specifying the index of the column of z to unscale.

`z` – A `double` matrix containing the values to be unscaled.

Decode

`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

`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.

Encode

`virtual public void Encode(int columnIndex, double[,] x)`

Description

Scales a single column of a two dimensional array of values.

Indexing is zero-based.

Its *columnIndex*-th column is modified in place.

Parameters

`columnIndex` – An `int` specifying the index of the column of x to scale.

`x` – A `double` matrix containing the value to be scaled.

Encode

`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 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.

GetBounds

`virtual public double[] GetBounds()`

Description

Retrieves bounds used during bounded scaling.

i	result[b]
0	`realMin`. Lowest expected value in the data to be filtered.
1	`realMax`. Largest expected value in the data to be filtered.
2	`targetMin`. Lowest allowed value in the filtered data.
3	`targetMax`. Largest allowed value in the filtered data.

Returns

A `double` array of length 4 containing the bounds.

SetBounds

`virtual public void SetBounds(double realMin, double realMax, double targetMin, double targetMax)`

Description

Sets bounds to be used during bounded scaling and unscaling.

This method is normally called prior to calls to Encode (p. 1106) or Decode (p. 1106). Otherwise the default bounds are $realMin = 0$, $realMax = 1$, $targetMin = 0$, and $targetMax = 1$. These bounds are ignored for unbounded scaling.

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.

Description

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:

Argument	Interpretation
`realMin`	The lowest value expected in `x`.
`realMax`	The largest value expected in `x`.
`targetMin`	The lower bound for the values in the scaled data.
`targetMax`	The upper bound for the values in the scaled data.

The scale limits are set using the method SetBounds (p. 1107).

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:

$$z = r(x - realMin) + targetMin,$$

where

$$r = \frac{targetMax - targetMin}{realMax - realMin}.$$

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 MAD is the Mean Absolute Deviation

$$\text{MAD} = median\{\mid x - median\{x\} \mid\}$$

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. 1106) is called then an unscaling operation is conducted by inverting using:

$$x = \frac{(z - targetMin)}{r} + realMin.$$

Unbounded z-score Scaling

If *scalingMethod* is `UnboundedZScoreMeanStdev` or `UnboundedZScoreMedianMAD`, then a scaling operation is conducted using the z-score calculation:

$$z = \frac{(x - center)}{spread},$$

If *scalingMethod* is `UnboundedZScoreMeanStdev` then Center (p. 1104) is set equal to the arithmetic average $\bar{x}$ of x, and Spread (p. 1104) 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:

$$x = spread \cdot z + center.$$

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:

$$z = \frac{r \cdot (x - center)}{spread} - r \cdot realMin + targetMin.$$

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:

$$x = \frac{spread \cdot (z - targetMin)}{r} + spread \cdot realMin + center$$

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

Variable	Method	Description
X_0	0	No Scaling
X_1	4	Bounded Z-score scaling using the mean and standard deviation of X_1
X_2	5	Bounded Z-score scaling using the median and MAD of X_2

The bounds, measuresof center and spreadfor $\mathbf{X_1}$ and $\mathbf{X_2}$ are:

Scaling Limits and Measures of Center and Spread

Variable	Real Limits	Target Limits	Measure of Center	Measure of Spread
X_1	(-6, +6)	(-3, +3)	3.4 (Mean)	1.7421 (Std. Dev.)
X_2	(-3, +3)	(-3, +3)	2.4 (Median)	1.3343(MAD/0.6745)

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 $\mathbf{x_1[0]=3.5}$ yields the following calculations:

For $\mathbf{x_1[0]}$, the scale factor is calculated using the real and target limits in the above table:

$\mathbf{r}$ = (3-(-3))/(6-(-6)) = 0.5

The z-score for $\mathbf{x_1[0]}$ is calculated using the measures of center and spread:

$\mathbf{z_1[0]}$ = (3.5 - 3.4)/1.7421 = 0.057402

Since method=4 is used for $\mathbf{x_1}$, this z-score is bounded (scaled) using the real and target limits:

$\mathbf{z_1(bounded)}$ = $\mathbf{r}$($\mathbf{z_1[0]}$) - r(realMin) + (targetMin)
= 0.5(0.057402) - 0.5(-6) + (-3) = 0.029

Thecalculations for $\mathbf{x_2[0]}$ are nearly identical, except that since method=5 for $\mathbf{x_2}$, the median and MAD replace the mean and standard deviation used to calculate $\mathbf{z_1(bounded):}$

$\mathbf{r}$ = (3-(-3))/(3-(-3)) = 1,

$\mathbf{z_2[0]}$ = (3.1 - 2.4)/1.3343 = 0.525, and

$\mathbf{z_2(bounded)}$ = $\mathbf{r}$($\mathbf{z_2[0]}$) - r(realMin) + (targetMin)
= 1(0.525) - 1(-3) + (-3) = 0.525

```
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl.DataMining.Neural;

public class ScaleFilterEx1
{
```

```
[STAThread]
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] + ", ");
   System.Console.Out.WriteLine(x0[4] + "}");
   // Display summary statistics for X1
   System.Console.Out.Write("\nX1 = {");
   for (int i = 0; i < 4; i++)
      System.Console.Out.Write(x1[i] + ", ");
   System.Console.Out.WriteLine(x1[4] + "}");
   System.Console.Out.WriteLine("X1 Mean:       " + scaleFilter[1].Center);
   System.Console.Out.WriteLine("X1 Std. Dev.:  " + scaleFilter[1].Spread);
   // Display summary statistics for X2
   System.Console.Out.Write("\nX2 = {");
   for (int i = 0; i < 4; i++)
      System.Console.Out.Write(x2[i] + ", ");
   System.Console.Out.WriteLine(x2[4] + "}");
   System.Console.Out.WriteLine("X2 Median:     " + scaleFilter[2].Center);
   System.Console.Out.WriteLine("X2 MAD/0.6745: " + scaleFilter[2].Spread);
   System.Console.Out.WriteLine("");
   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);
   pm.SetTitle("Decoded Z0");
```

```
        pm.Print(z0);
        pm.SetTitle("Decoded Z1");
        pm.Print(z1);
        pm.SetTitle("Decoded Z2");
        pm.Print(z2);
    }
}
```

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   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
0   0.524603139041397
1  -0.674489750196082
2  -2.92278891751635
3   0
4   1.34897950039216

Decoded Z0
    0
0   1.2
1   0
2  -1.4
3   1.5
4   3.2
```

```
Decoded Z1
    0
0  3.5
1  2.4
2  4.4
3  5.6
4  1.1

Decoded Z2
    0
0   3.1
1   1.5
2  -1.5
3   2.4
4   4.2
```

ScaleFilter.ScalingMethod Enumeration

Summary

Scaling Method

`public enumeration Imsl.DataMining.Neural.ScaleFilter.ScalingMethod`

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

Summary

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.

```
public class Imsl.DataMining.Neural.UnsupervisedNominalFilter
```

Property

NumberOfClasses

```
virtual public int NumberOfClasses {get; }
```

Description

The number of classes in the nominal variable.

Constructor

UnsupervisedNominalFilter

`public UnsupervisedNominalFilter(int nClasses)`

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 matrix representing the binary encoded columns of the nominal variable.

This is the inverse of the Encode (p. 1116) method.

Parameter

`z` – An `int` matrix containing the data to be decoded.

Returns

An `int` array containing the decoded data.

Decode

`virtual public int Decode(int[] z)`

Description

Decodes a binary encoded array into its nominal category.

This is the inverse of the Encode (p. 1116) method.

Parameter

`z` – An `int` array containing the data to be decoded.

Returns

An `int` containing the number associated with the category encoded in z.

Encode

`virtual public int[] Encode(int x)`

Description

Apply forward encoding to a value.

Class number must be in the range 1 to *nClasses*.

Parameter

`x` – An `int` containing the value to be encoding.

Returns

An `int` array containing the encoded data.

Encode

`virtual public int[,] Encode(int[] x)`

Description

Encodes class data prior to its use in neural network training.

Class number must be in the range 1 to *nClasses*.

Parameter

`x` – An `int` array containing the data to be encoded.

Returns

An `int` matrix containing the encoded data.

Description

Binary Encoding

Method Encode (p. 1116) 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

$$z = \begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

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. 1115). 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.

Example: UnsupervisedNominalFilter

In this example a data set with 7 observations and 3 classes is filtered.

```
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl.DataMining.Neural;

public class UnsupervisedNominalFilterEx1
{
   [STAThread]
   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

Summary

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.

`public class Imsl.DataMining.Neural.UnsupervisedOrdinalFilter`

Properties

NumberOfClasses

`virtual public int NumberOfClasses {get; }`

Description

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.

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.

Constructor

UnsupervisedOrdinalFilter

```
public UnsupervisedOrdinalFilter(int nClasses,
  Imsl.DataMining.Neural.UnsupervisedOrdinalFilter.TransformMethod transform)
```

Description

Constructor for `UnsupervisedOrdinalFilter`.

Values for Transform (p. 1119) are: TransformMethod.None (p. 1123), TransformMethod.Sqrt (p. 1123), TransformMethod.AsinSqrt (p. 1123)

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.

Methods

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.

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.

Encode

```
virtual public double Encode(int x)
```

Description

Encodes an ordinal category.

x must be an integer between 1 and *nClasses*.

Parameter

`x` – An `int` containing the ordinal category.

Returns

A `double` containing the encoded value, a transformed cumulative percentage.

Encode

```
virtual public double[] Encode(int[] x)
```

Description

Encodes an array of ordinal categories into an array of transformed percentages.

Categories must be numbered from 1 to *nClasses*.

Parameter

`x` – An `int` array containing the categories for the ordinal variable.

Returns

A `double` array of the transformed percentages.

Description

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. 1120) 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 *nClasses* `=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.

Ordinal Class	Frequency	Cumulative Percentage
1	3	30%
2	2	50%
3	3	80%
4	2	100%

Classes in x must be numbered from 1 to *nClasses*.

The values returned from encoding or decoding depend upon the setting of Transform (p. 1119). In this example, if the filter was constructed with `Transform = TransformMethod.None`, then the method `Encode` will return

$$z[] = \{50, 30, 80, 100, 50, 100, 30, 30, 80, 80\}.$$

If the filter was constructed with `Transform = TransformMethod.Sqrt`, then the square root of these values is returned; i.e.,

$$z[i] = \sqrt{\frac{z[i]}{100}}$$

$$z[] = \{0.71, 0.55, 0.89, 1.0, 0.71, 1.0, 0.55, 0.55, 0.89, 0.89\};$$

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.

Example: UnsupervisedOrdinalFilter

In this example a data set with 10 observations and 4 classes is filtered.

```
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl.DataMining.Neural;

public class UnsupervisedOrdinalFilterEx1
{
   [STAThread]
   public static void  Main(System.String[] args)
   {
      int nClasses = 4;
      UnsupervisedOrdinalFilter filter = new UnsupervisedOrdinalFilter(nClasses, UnsupervisedOrdinalFilter.Transfor
      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.SetNoRowLabels();
      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
3
3
```

UnsupervisedOrdinalFilter.TransformMethod Enumeration

Summary

Transform type

```
public enumeration
Imsl.DataMining.Neural.UnsupervisedOrdinalFilter.TransformMethod
```

Fields

`AsinSqrt`

```
public Imsl.DataMining.Neural.UnsupervisedOrdinalFilter.TransformMethod
  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`

```
public Imsl.DataMining.Neural.UnsupervisedOrdinalFilter.TransformMethod Sqrt
```

Description

Flag to indicate the square root transform will be applied to the percentages.

TimeSeriesFilter Class

Summary

Converts time series data to a lagged format used as input to a neural network.

`public class Imsl.DataMining.Neural.TimeSeriesFilter`

Constructor

TimeSeriesFilter

`public TimeSeriesFilter()`

Description

Constructor for `TimeSeriesClassFilter`.

Method

ComputeLags

`virtual public double[,] ComputeLags(int nLags, double[,] x)`

Description

Lags time series data to a format used for input to a neural network.

nLags must be greater than 0.

It is assumed that x is sorted in descending chronological order.

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.

Description

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. 1124), each column of the input matrix, x, is transformed into ($nLags$+1) columns by creating a column for $lags = 0, 1, \ldots nLags$.

The output data array, z, 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 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:

$$z = \begin{bmatrix} 1 & 6 & 2 & 7 \\ 2 & 7 & 3 & 8 \\ 3 & 8 & 4 & 9 \\ 4 & 9 & 5 & 10 \end{bmatrix}$$

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:

$$z = \begin{bmatrix} 1 & 6 & 2 & 7 & 3 & 8 \\ 2 & 7 & 3 & 8 & 4 & 9 \\ 3 & 8 & 4 & 9 & 5 & 10 \end{bmatrix}$$

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.

```
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl.DataMining.Neural;
```

```
public class TimeSeriesFilterEx1
{
   [STAThread]
   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

Summary

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.

`public class Imsl.DataMining.Neural.TimeSeriesClassFilter`

Constructor

TimeSeriesClassFilter

`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

```
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.

Every lag must be non-negative.

The i-th element of $iClass$ is equal to the class associated with the i-th element of x. $iClass$ 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.

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 `lags[i]`. The number of rows is equal to the length of `x`.

Description

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. 1127), 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

$$x^T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\},$$
$$iClass^T = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1\},$$

and

$$lag^T = \{0, 2\}$$

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]` $= 2$.

$$z = \begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \\ 4 & 6 \\ 5 & 7 \\ 6 & 8 \\ 7 & 9 \\ 8 & 10 \\ 9 & NaN \\ 10 & NaN \end{bmatrix}$$

On the other hand, if the data were organized into two classes with

$$iClass^T = \{1, 1, 1, 1, 1, 2, 2, 2, 2, 2\},$$

then *nClasses* is 2, and z is still a 2 by 10 matrix, but with the following values:

$$z = \left[\begin{array}{cc} 1 & 3 \\ 2 & 4 \\ 3 & 5 \\ 4 & NaN \\ 5 & NaN \\ \hline 6 & 8 \\ 7 & 9 \\ 8 & 10 \\ 9 & NaN \\ 10 & NaN \end{array}\right]$$

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.

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=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=NaN, T_1=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.

```
using System;
using Imsl.Stat;
using Imsl.Math;
using Imsl.DataMining.Neural;

public class TimeSeriesClassFilterEx1
{
   private static int nClasses = 2;
   private static int nObs = 10;
   private static int nLags = 4;
   [STAThread]
   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};
      System.String[] colLabels = new System.String[]{"Class", "Time", "Lag=0",
         "Lag=1", "Lag=2", "Lag=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

```
               Lagged data
Class  Time  Lag=0  Lag=1  Lag=2  Lag=3
1      3      1      2      3      4
1      2      2      3      4      NaN
1      1      3      4      NaN    NaN
1      0      4      NaN    NaN    NaN
2      5      5      6      7      8
2      4      6      7      8      9
2      3      7      8      9     10
2      2      8      9     10      NaN
2      1      9     10      NaN    NaN
2      0     10      NaN    NaN    NaN
```

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:

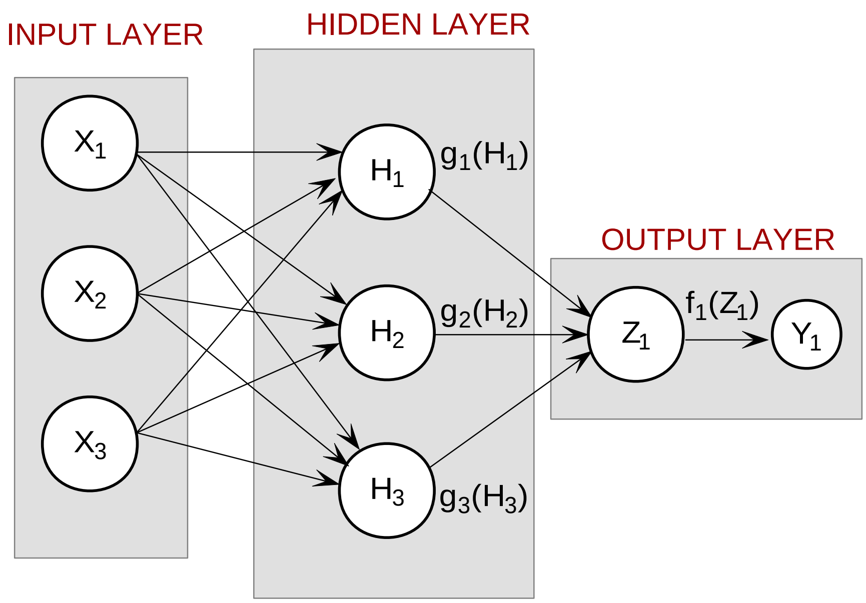


Figure 9. An example 2-layer Feed Forward Neural Network

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 funtion 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 interchangable. 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

```
using System;
using Imsl.DataMining.Neural;
using Imsl.Math;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
//*****************************************************************************
// 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.                                         *
// 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";
   // INPUT FILES
   // Continuous input attributes file.  File contains Lags 1-3 of series
   private static System.String contFileName = "continuous.txt";
   // Continuous network targets file.  File contains the original series
   private static System.String outputFileName = "output.txt";
   // ***********************************************************************
   // Data Preprocessing Settings                                           *
   // ***********************************************************************
   private static double lowerDataLimit = - 105000; // lower scale limit
   private static double upperDataLimit = 25000000; // upper scale limit
```

```
// indicator
// ***************************************************************************
// Time Parameters for Tracking Training Time                               *
// ***************************************************************************
private static int startTime;
// ***************************************************************************
// Error Message Encoding for Stage II Trainer - Quasi-Newton Trainer       *
// ***************************************************************************
// Note: For the Epoch Trainer, the error status returned is the status for*
// the Stage II trainer, unless Stage II training is not used.              *
// ***************************************************************************
private static System.String errorMsg = "";
// Error Status Messages for the Quasi-Newton Trainer
private static System.String errorMsg0 = "--> Network Training";
private static System.String errorMsg1 =
   "--> The last global step failed to locate a lower point than the\n" +
   "current error value.  The current solution may be an approximate\n" +
   "solution and no more accuracy is possible, or the step tolerance\n" +
   "may be too large.";
private static System.String errorMsg2 =
   "--> Relative function convergence;  both both the actual and \n" +
   "predicted relative reductions in the error function are less than\n" +
   "or equal to the relative fu nction convergence tolerance.";
private static System.String errorMsg3 =
   "--> Scaled step tolerance satisfied;  the current solution may be\n" +
   "an approximate local solution, or the algorithm is making very slow\n" +
   "progress and is not near a solution, or the step tolerance is too big.";
private static System.String errorMsg4 =
   "--> Quasi-Newton Trainer threw a \n" +
   "MinUnconMultiVar.FalseConvergenceException.";
private static System.String errorMsg5 =
   "--> Quasi-Newton Trainer threw a \n" +
   "MinUnconMultiVar.MaxIterationsException.";
private static System.String errorMsg6 =
   "--> Quasi-Newton Trainer threw a \n" +
      "MinUnconMultiVar.UnboundedBelowException.";
// ***************************************************************************
// MAIN                                                                     *
// ***************************************************************************
[STAThread]
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 tragets
   int i, j, m = 0; // Array indicies
   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[]{0};
   //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;
System.Console.Out.WriteLine("--> Starting Data Preprocessing at: " +
   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:        " +
   nOutputs);
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++;
}
System.Console.Out.WriteLine("--> Number of Missing Observations:     "
   + 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 = 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                                *
// ******************************************************************
System.Console.Out.WriteLine("--> Training 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;
System.Console.Out.WriteLine("--> Network Training Completed at: " +
   currentTimeNow.ToString());
double duration = (double) (currentTimeNow - startTime) / 1000.0;
System.Console.Out.WriteLine("--> Training Time: " + duration +
   " seconds");

// ***********************************************************************
// 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]);
```

```
System.Console.Out.WriteLine("--> Laplacian Error:          " +
   (float)stats[2]);
System.Console.Out.WriteLine("--> Scaled Laplacian Error:   " +
   (float)stats[3]);
System.Console.Out.WriteLine("--> Largest Absolute Residual: " +
   (float)stats[4]);
System.Console.Out.WriteLine(
   "**********************************************");
System.Console.Out.WriteLine("");
// **********************************************************************
// OBTAIN AND DISPLAY NETWORK WEIGHTS AND GRADIENTS                     *
// **********************************************************************
System.Console.Out.WriteLine("--> Getting Network Weights and Gradients");
// 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("--> Network Weights and Gradients:");
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);
pm.SetTitle("Weights and Gradients");
pm.Print(mf, printMatrix);

System.Console.Out.WriteLine(
   "**********************************************");
// **********************************************************************
// SAVE THE TRAINED NETWORK BY SAVING THE SERIALIZED NETWORK OBJECT     *
// **********************************************************************
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 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)
    {
       System.Console.Out.WriteLine("--> ERROR READING 2nd LINE OF FILE: "
          + 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  -> 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;
}
// **************************************************************************
// Create Stage I/Stage II Trainer                                          *
// **************************************************************************
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.
System.Console.Out.WriteLine("    --> Creating Epoch Trainer");
if (s1.Equals(QuasiNewton))
{
   // Setup stage I quasi-newton trainer
   stage1Trainer = new QuasiNewtonTrainer();
   //stage1Trainer.setMaximumStepsize(maxStepSize);
   stage1Trainer.MaximumTrainingIterations = stage1Iterations;
   stage1Trainer.StepTolerance = stage1StepTolerance;
   if (s2.Equals(QuasiNewton))
   {
      stage2Trainer = new QuasiNewtonTrainer();
      //stage2Trainer.setMaximumStepsize(maxStepSize);
      stage2Trainer.MaximumTrainingIterations = stage2Iterations;
      epoch = new EpochTrainer(stage1Trainer, stage2Trainer);
   }
   else
   {
      if (s2.Equals(LeastSquares))
      {
         stage2LS = new LeastSquaresTrainer();
         stage2LS.InitialTrustRegion = 1.0e-3;
         //stage2LS.setMaximumStepsize(maxStepSize);
         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;
   //stage1LS.setMaximumStepsize(maxStepSize);
   if (s2.Equals(QuasiNewton))
   {
      stage2Trainer = new QuasiNewtonTrainer();
      //stage2Trainer.setMaximumStepsize(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);
   System.Console.Out.WriteLine("    --> Stage I Step Tolerance:      " +
      stage1StepTolerance);
   System.Console.Out.WriteLine("    --> Stage I Relative Tolerance:  " +
      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 =
      DateTime.Now.Hour * 60 * 60 * 1000 +
      DateTime.Now.Minute * 60 * 1000 +
      DateTime.Now.Second * 1000 +
      DateTime.Now.Millisecond;
   System.Console.Out.WriteLine("--> Starting Network Training at " +
      currentTimeNow.ToString());
   // Return Stage I/Stage II trainer
   return epoch;
}

// ***************************************************************************
```

```
   // WRITE SERIALIZED OBJECT 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 NeuralNetworkEx1()
   {
      hiddenLayerActivation = Imsl.DataMining.Neural.Activation.LogisticTable;
      outputLayerActivation = Imsl.DataMining.Neural.Activation.Linear;
   }
}
// ************************************************************************
```

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
    --> Stage II Relative Tolerance: 1E-11
    --> Stage II Step Size:          DEFAULT
    --> Stage II Trace:              True
--> 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

--> Network Weights and Gradients:
**********************************************
             Weights and Gradients
         Weight              Gradient
-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
-258.811023180973     -2.81816574426092E-05
-394.380943852438     -0.000125916731945308
  -0.356726621727131  -5.25467092773031E-07
 394.428311058654     -2.70798222353788E-05
 422.855858784789     -1.40339989032276E-06
  -1.01024906891467   -8.54119524733673E-07
 422.854960914701      3.37315953950526E-08
  91.0301743864326    -0.000555459860183764
   0.672279284955327  -3.11957565142863E-06
 -91.0431760187523    -0.000120208750794691
-422.186774012951     -1.36686903761535E-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: 84904271
--> 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 84925490
--> 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: 86184862
--> Training Time: 1259.372 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

--> Network Weights and Gradients:
**********************************************
             Weights and Gradients
         Weight              Gradient
-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
-258.811023180973     -2.81816574426092E-05
-394.380943852438     -0.000125916731945308
  -0.356726621727131  -5.25467092773031E-07
 394.428311058654     -2.70798222353788E-05
 422.855858784789     -1.40339989032276E-06
  -1.01024906891467   -8.54119524733673E-07
 422.854960914701      3.37315953950526E-08
  91.0301743864326    -0.000555459860183764
   0.672279284955327  -3.11957565142863E-06
 -91.0431760187523    -0.000120208750794691
```

```
-422.186774012951     -1.36686903761535E-06

**********************************************

--> 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

Summary

Neural network base class.

`public class Imsl.DataMining.Neural.Network`

Properties

InputLayer

`abstract public Imsl.DataMining.Neural.InputLayer InputLayer {get; }`

Description

The `InputLayer` object.

Links

`abstract public Imsl.DataMining.Neural.Link[] Links {get; }`

Description

An array containing the `Link` objects in the `Network`.

NumberOfInputs

`abstract public int NumberOfInputs {get; }`

Description

The number of `Network` inputs.

NumberOfLinks

```
abstract public int NumberOfLinks {get; }
```

Description

The number of `Network Links` among the `nodes`.

NumberOfOutputs

```
abstract public int NumberOfOutputs {get; }
```

Description

The number of `Network` output Perceptrons (p. 1026).

NumberOfWeights

```
abstract public int NumberOfWeights {get; }
```

Description

The number of Weights (p. 1029) in the `Network`.

OutputLayer

```
abstract public Imsl.DataMining.Neural.OutputLayer OutputLayer {get; }
```

Description

The `OutputLayer`.

Perceptrons

```
abstract public Imsl.DataMining.Neural.Perceptron[] Perceptrons {get; }
```

Description

An array containing the `Perceptrons` in the `Network`.

Weights

```
abstract public double[] Weights {get; set; }
```

Description

The Weights (p. 1029).

Constructor

Network

```
public Network()
```

Description

Default constructor for `Network`.

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.

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 $\bar{y} = \sum_i y_i/NC$, where N is the number of observations and C is the number of classes per observation.

Index	Name	Formula
0	SSE	$\frac{1}{2}\sum_i (y_i - \hat{y}_i)^2$
1	RMS	$\frac{\sum_i (y_i-\hat{y}_i)^2}{\sum_i (y_i-\bar{y}_i)}$
2	Laplacian	$\sum_i \vert y_i - \hat{y}_i\vert$
3	Scaled Laplacian	$\frac{\sum_i \vert y_i-\hat{y}_i\vert}{\sum_i \vert y_i-\bar{y}_i\vert}$
4	Max residual	$\max_i \vert y_i - \hat{y}_i\vert$

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.

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. 1026). Its length is equal to the number of output `Perceptron`s.

GetForecastGradient

```
abstract public double[,] GetForecastGradient(double[] x)
```

Description

Returns the derivatives of the outputs with respect to the Weights (p. 1029) 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:

$y = 10*X_1 + 20*X_2 + 30*X_3 + 2.0*X_4$, where

X_1-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:

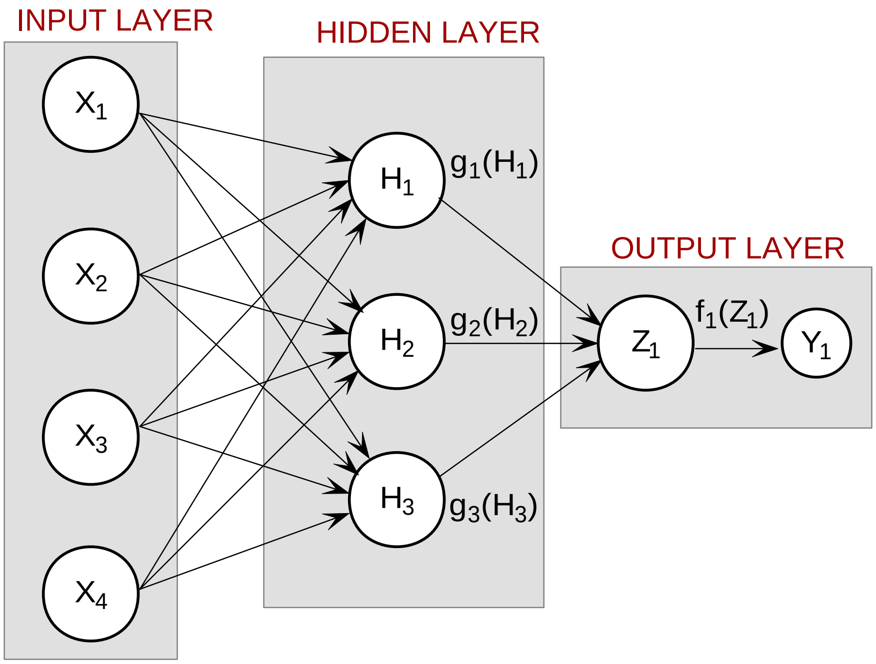


Figure 10. An example 2-layer Feed Forward Neural Network with 4 Inputs

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;
using System.Runtime.Serialization.Formatters.Binary;
//*****************************************************************************
// 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
   // **********************************************************************
   [STAThread]
   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]);
System.Console.Out.WriteLine("--> Laplacian Error:           " +
   (float)stats[2]);
System.Console.Out.WriteLine("--> Scaled Laplacian Error:    " +
   (float)stats[3]);
System.Console.Out.WriteLine("--> Largest Absolute Residual: " +
   (float)stats[4]);
System.Console.Out.WriteLine(
   "***********************************************");
System.Console.Out.WriteLine("");
// **********************************************************************
// OBTAIN AND DISPLAY NETWORK WEIGHTS AND GRADIENTS
// **********************************************************************
System.Console.Out.WriteLine("--> Getting Network Information");
// Get weights
weight = network.Weights;
// Get number of weights = number of gradients
int nWeights = network.NumberOfWeights;
// Obtain Gradient Vector
gradient = trainer.ErrorGradient;
// Print Network Weights and Gradients
System.Console.Out.WriteLine(" ");
System.Console.Out.WriteLine("--> Network Weights and Gradients:");
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.Write("--> X1=" + (int) x[i,0] + " X2=" +
            (int) x[i,1] + " X3=" + (int) x[i,2] + " | X4=" + x[i,3]);
         System.Console.Out.WriteLine(" | 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;
   }
}
```

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
--> 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[3]=1.616918 g[3]=6.182035E-08
w[4]=1.616918 g[4]=6.182035E-08
w[5]=1.616918 g[5]=6.182035E-08
w[6]=4.725622 g[6]=-5.273856E-08
w[7]=4.725622 g[7]=-5.273856E-08
w[8]=4.725622 g[8]=-5.273856E-08
w[9]=6.217407 g[9]=-8.733E-10
w[10]=6.217407 g[10]=-8.733E-10
```

```
w[11]=6.217407 g[11]=-8.733E-10
w[12]=1.072258 g[12]=-1.690978E-07
w[13]=1.072258 g[13]=-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 25: Miscellaneous

Types

Warning Class

Summary

Handles warning messages.

`public class Imsl.Warning`

Properties

WarningObject

`static public Imsl.WarningObject WarningObject {get; set; }`

Description

The `WarningObject` allows warning errors to be handled in a more custom fashion.

`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.

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. 1157) class.

Method

Print

```
static public void Print(Object source, string bundleName, string key,
  Object[] arg)
```

Description

Issues a warning message.

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.

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.

Description

This class maintains a single, private, `WarningObject` that actually displays the warning messages.

WarningObject Class

Summary

Handles warning messages.

```
public class Imsl.WarningObject
```

Property

Writer

```
public System.IO.TextWriter Writer {get; set; }
```

Description

Reassigns the writer.

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.

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.

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".

`key` – A `String` which specifies the warning message in the resource.

`arg` – A `Object` which specifies arguments used to format the message.

IMSLException Class

Summary

Signals that a mathematical exception has occurred.

`public class Imsl.IMSLException : ApplicationException : ISerializable`

Constructors

IMSLException

`IMSLException()`

Description

Constructs an `IMSLException` with no detail message.

A detail message is a `String` that describes this particular exception.

IMSLException

`IMSLException(string s)`

Description

Constructs an `IMSLException` with the specified detail message.

A detail message is a `String` that describes this particular exception.

Parameter

`s` – A `String` which specifies the detail message.

IMSLException

`IMSLException(string namespaceName, string key, Object[] arguments)`

Description

Constructs an `IMSLException` with the specified detail message.

The error message `String` is in a resource bundle, ErrorMessages.

Parameters

`namespaceName` – A `String` which specifies the namespace containing the ErrorMessages resource bundle.

`key` – 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.

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,
 System.Runtime.Serialization.StreamingContext context)
```

Description

Constructs an `IMSLException` with the serialized data.

Parameters

`info` – The `Object` that holds the serialized object data.

`context` – The contextual information about the source or destination.

Chapter 26: Exceptions

Types

BadInitialGuessException Class

Summary

Penalty function point infeasible for original problem. Try new initial guess.

```
public class Imsl.Math.BadInitialGuessException : IMSLException :
ISerializable
```

Constructors

BadInitialGuessException

```
public BadInitialGuessException()
```

Description

Penalty function point infeasible for original problem. Try new initial guess.

BadInitialGuessException

```
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

```
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

```
BadInitialGuessException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

BoundsInconsistentException Class

Summary

The bounds given are inconsistent.

```
public class Imsl.Math.BoundsInconsistentException : IMSLException :
ISerializable
```

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

```
BoundsInconsistentException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Constraint evaluation returns an error with current point.

```
public class Imsl.Math.ConstraintEvaluationException : IMSLException :
ISerializable
```

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

```
ConstraintEvaluationException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The equality constraints are inconsistent.

```
public class Imsl.Math.ConstraintsInconsistentException : IMSLException :
ISerializable
```

Constructors

ConstraintsInconsistentException

```
public ConstraintsInconsistentException()
```

Description

The equality constraints are inconsistent.

ConstraintsInconsistentException

```
public ConstraintsInconsistentException(string message)
```

Description

The equality constraints are inconsistent.

Parameter

`message` – The error message that explains the reason for the exception.

ConstraintsInconsistentException

```
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.

ConstraintsInconsistentException

```
ConstraintsInconsistentException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

Description

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

Summary

No vector x satisfies all of the constraints.

```
public class Imsl.Math.ConstraintsNotSatisfiedException :  IMSLException :
ISerializable
```

Constructors

ConstraintsNotSatisfiedException

```
public ConstraintsNotSatisfiedException()
```

Description

No vector x satisfies all of the constraints.

ConstraintsNotSatisfiedException

```
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

```
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

```
ConstraintsNotSatisfiedException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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.

DidNotConvergeException Class

Summary

Maximum number of iterations exceeded.

```
public class Imsl.Math.DidNotConvergeException : IMSLException :
ISerializable
```

Constructors

DidNotConvergeException

```
public DidNotConvergeException()
```

Description

Maximum number of iterations exceeded.

DidNotConvergeException

```
public DidNotConvergeException(int maximumNumberOfIterations)
```

Description

Maximum number of iterations exceeded.

Parameter

`maximumNumberOfIterations` – Maximum number of iterations allowed exceeded argument.

DidNotConvergeException

`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

`public DidNotConvergeException(string message)`

Description

Maximum number of iterations exceeded.

Parameter

`message` – The error message that explains the reason for the exception.

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

```
DidNotConvergeException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The variables are determined by the equality constraints.

```
public class Imsl.Math.EqualityConstraintsException : IMSLException :
ISerializable
```

Constructors

EqualityConstraintsException

```
public EqualityConstraintsException()
```

Description

The variables are determined by the equality constraints.

EqualityConstraintsException

```
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.

EqualityConstraintsException

```
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.

EqualityConstraintsException

```
EqualityConstraintsException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

FalseConvergenceException Class

Summary

False convergence, the iterates appear to be converging to a noncritical point.

```
public class Imsl.Math.FalseConvergenceException :  IMSLException :
ISerializable
```

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

```
FalseConvergenceException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Problem is singular or ill-conditioned.

```
public class Imsl.Math.IllConditionedException : IMSLException :
ISerializable
```

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, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Inconsistent system.

```
public class Imsl.Math.InconsistentSystemException : IMSLException :
ISerializable
```

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

```
InconsistentSystemException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

Description

Inconsistent system.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

LimitingAccuracyException Class

Summary

Limiting accuracy reached for a singular problem.

```
public class Imsl.Math.LimitingAccuracyException :  IMSLException :
ISerializable
```

Constructors

LimitingAccuracyException

```
public LimitingAccuracyException()
```

Description

Limiting accuracy reached for a singular problem.

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

```
LimitingAccuracyException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Working set gradients are linearly dependent.

```
public class Imsl.Math.LinearlyDependentGradientsException :  IMSLException :
ISerializable
```

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

```
LinearlyDependentGradientsException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Maximum number of iterations exceeded.

```
public class Imsl.Math.MaxIterationsException : IMSLException : ISerializable
```

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

```
MaxIterationsException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Maximum number of steps allowed exceeded.

```
public class Imsl.Math.MaxNumberStepsAllowedException :  IMSLException :
ISerializable
```

Constructors

MaxNumberStepsAllowedException

```
public MaxNumberStepsAllowedException()
```

Description

Maximum number of steps allowed exceeded.

MaxNumberStepsAllowedException

```
public MaxNumberStepsAllowedException(int maxSteps)
```

Description

Maximum number of steps allowed exceeded.

Parameter

`maxSteps` – Maximum number of steps allowed.

MaxNumberStepsAllowedException

```
public MaxNumberStepsAllowedException(string message)
```

Description

Maximum number of steps allowed exceeded.

Parameter

`message` – The error message that explains the reason for the exception.

MaxNumberStepsAllowedException

```
public MaxNumberStepsAllowedException(string s, System.Exception exception)
```

Description

Maximum number of steps allowed 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.

MaxNumberStepsAllowedException

```
MaxNumberStepsAllowedException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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.

NoAcceptableStepsizeException Class

Summary

No acceptable stepsize in [SIGMA,SIGLA].

```
public class Imsl.Math.NoAcceptableStepsizeException :  IMSLException :
ISerializable
```

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

```
NoAcceptableStepsizeException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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.

NotSPDException Class

Summary

The matrix is not symmetric, positive definite.

`public class Imsl.Math.NotSPDException : IMSLException : ISerializable`

Constructors

NotSPDException

`public NotSPDException()`

Description

The matrix is not symmetric, positive definite.

NotSPDException

```
public NotSPDException(string message)
```

Description

The matrix is not symmetric, positive definite.

Parameter

`message` – The error message that explains the reason for the exception.

NotSPDException

```
public NotSPDException(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.

NotSPDException

```
NotSPDException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Numerical difficulty occurred.

```
public class Imsl.Math.NumericDifficultyException :  IMSLException :
ISerializable
```

Constructors

NumericDifficultyException

```
public NumericDifficultyException()
```

Description

Numerical difficulty occurred.

NumericDifficultyException

```
public NumericDifficultyException(string message)
```

Description

Numerical difficulty occurred.

Parameter

`message` – The error message that explains the reason for the exception.

NumericDifficultyException

```
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.

NumericDifficultyException

```
NumericDifficultyException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Objective evaluation returns an error with current point.

```
public class Imsl.Math.ObjectiveEvaluationException : IMSLException :
ISerializable
```

Constructors

ObjectiveEvaluationException

```
public ObjectiveEvaluationException()
```

Description

Objective evaluation returns an error with current point.

ObjectiveEvaluationException

```
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

```
public ObjectiveEvaluationException(string s, System.Exception exception)
```

Description

Objective 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.

ObjectiveEvaluationException

```
ObjectiveEvaluationException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Penalty function point infeasible.

```
public class Imsl.Math.PenaltyFunctionPointInfeasibleException : IMSLException
: ISerializable
```

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

```
PenaltyFunctionPointInfeasibleException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

Description

Penalty function point infeasible.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

ProblemInfeasibleException Class

Summary

The problem is not feasible. The constraints are inconsistent.

```
public class Imsl.Math.ProblemInfeasibleException : IMSLException :
ISerializable
```

Constructors

ProblemInfeasibleException

```
public ProblemInfeasibleException()
```

Description

The problem is not feasible. The constraints are inconsistent.

ProblemInfeasibleException

```
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.

ProblemInfeasibleException

```
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.

ProblemInfeasibleException

```
ProblemInfeasibleException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The problem is unbounded.

```
public class Imsl.Math.ProblemUnboundedException :  IMSLException :
ISerializable
```

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, System.Runtime.Serialization.StreamingContext context)
```

Description

The problem is unbounded.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

QPInfeasibleException Class

Summary

QP problem seemingly infeasible.

```
public class Imsl.Math.QPInfeasibleException : IMSLException : ISerializable
```

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

```
QPInfeasibleException(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context)
```

Description

QP problem seemingly infeasible.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

SingularException Class

Summary

Problem is singular.

`public class Imsl.Math.SingularException : IMSLException : ISerializable`

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

`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.

SingularException

```
SingularException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The matrix is singular.

```
public class Imsl.Math.SingularMatrixException :  IMSLException :
ISerializable
```

Constructors

SingularMatrixException

```
public SingularMatrixException()
```

Description

The matrix is singular.

SingularMatrixException

```
public SingularMatrixException(string message)
```

Description

The matrix is singular.

Parameter

`message` – The error message that explains the reason for the exception.

SingularMatrixException

```
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

```
SingularMatrixException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

Description

The matrix is singular.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

TerminationCriteriaNotSatisfiedException Class

Summary

Termination criteria are not satisfied.

```
public class Imsl.Math.TerminationCriteriaNotSatisfiedException :
IMSLException :  ISerializable
```

Constructors

TerminationCriteriaNotSatisfiedException

```
public TerminationCriteriaNotSatisfiedException(int numsm)
```

Description

Termination criteria are not satisfied.

Parameter

`numsm` – An `int`containing the criteria value.

TerminationCriteriaNotSatisfiedException

```
public TerminationCriteriaNotSatisfiedException()
```

Description

Termination criteria are not satisfied.

TerminationCriteriaNotSatisfiedException

```
public TerminationCriteriaNotSatisfiedException(string message)
```

Description

Termination criteria are not satisfied.

Parameter

`message` – The error message that explains the reason for the exception.

TerminationCriteriaNotSatisfiedException

```
public TerminationCriteriaNotSatisfiedException(string s, System.Exception
  exception)
```

Description

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

```
TerminationCriteriaNotSatisfiedException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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.

ToleranceTooSmallException Class

Summary

Tolerance is too small.

```
public class Imsl.Math.ToleranceTooSmallException :  IMSLException :
ISerializable
```

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.

`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.

ToleranceTooSmallException

```
ToleranceTooSmallException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Maximum number of iterations exceeded.

```
public class Imsl.Math.TooManyIterationsException : IMSLException :
ISerializable
```

Constructors

TooManyIterationsException

```
public TooManyIterationsException(int maximumNumberOfIterations)
```

Description

Maximum number of iterations exceeded.

Parameter

`maximumNumberOfIterations` – 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

```
TooManyIterationsException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

UnboundedBelowException Class

Summary

Five consecutive steps of the maximum allowable stepsize have been taken, either the function is unbounded below, or has a finite asymptote in some directionor the maximum allowable step size is too small.

```
public class Imsl.Math.UnboundedBelowException : IMSLException :
ISerializable
```

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 directionor 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 directionor 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 directionor 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, System.Runtime.Serialization.StreamingContext context)
```

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 directionor 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

Summary

The equality constraints and the bounds on the variables are found to be inconsistent.

```
public class Imsl.Math.VarBoundsInconsistentException : IMSLException :
ISerializable
```

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

```
VarBoundsInconsistentException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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.

WorkingSetSingularException Class

Summary

Working set is singular in dual extended QP.

```
public class Imsl.Math.WorkingSetSingularException : IMSLException :
ISerializable
```

Constructors

WorkingSetSingularException

```
public WorkingSetSingularException()
```

Description

Working set is singular in dual extended QP.

WorkingSetSingularException

```
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

```
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

```
WorkingSetSingularException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

There are no observations.

```
public class Imsl.Stat.AllDeletedException : IMSLException : ISerializable
```

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.

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

```
AllDeletedException(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context)
```

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

Summary

There are no observations.

```
public class Imsl.Stat.AllMissingException : IMSLException : ISerializable
```

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

`AllMissingException(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)`

Description

There are no observations.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

BadVarianceException Class

Summary

The input variance is not in the allowed range.

`public class Imsl.Stat.BadVarianceException : IMSLException : ISerializable`

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

```
BadVarianceException(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context)
```

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.

ClassificationVariableException Class

Summary

The ClassificationVariable vector has not been initialized.

`public class Imsl.Stat.ClassificationVariableException : IMSLException : ISerializable`

Constructors

ClassificationVariableException

`public ClassificationVariableException()`

Description

The ClassificationVariable vector has not been initialized.

ClassificationVariableException

`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

`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

```
ClassificationVariableException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The Classification Variable limit set by the user through `setUpperBound` has been exceeded.

```
public class Imsl.Stat.ClassificationVariableLimitException : IMSLException :
ISerializable
```

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)
```

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

```
ClassificationVariableLimitException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The number of distinct values for each Classification Variable must be greater than 1.

```
public class Imsl.Stat.ClassificationVariableValueException : IMSLException :
ISerializable
```

Constructors

ClassificationVariableValueException

```
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

```
ClassificationVariableValueException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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.

ClusterNoPointsException Class

Summary

There is a cluster with no points.

```
public class Imsl.Stat.ClusterNoPointsException : IMSLException :
ISerializable
```

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

```
ClusterNoPointsException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The equality constraints are inconsistent.

```
public class Imsl.Stat.ConstrInconsistentException : IMSLException :
ISerializable
```

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.

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.

ConstrInconsistentException

```
ConstrInconsistentException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

Description

The equality constraints are inconsistent.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

CovarianceSingularException Class

Summary

The variance-Covariance matrix is singular.

```
public class Imsl.Stat.CovarianceSingularException :  IMSLException :
ISerializable
```

Constructors

CovarianceSingularException

`public CovarianceSingularException()`

Description

The variance-Covariance matrix is singular.

CovarianceSingularException

`public CovarianceSingularException(int l)`

Description

The variance-Covariance matrix is singular.

Parameter

`l` – A `int` which specifies the population number.

CovarianceSingularException

`public CovarianceSingularException(string message)`

Description

The variance-Covariance matrix is singular.

Parameter

`message` – The error message that explains the reason for the exception.

CovarianceSingularException

`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

```
CovarianceSingularException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

CyclingIsOccurringException Class

Summary

Cycling is occurring.

`public class Imsl.Stat.CyclingIsOccurringException : IMSLException : ISerializable`

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

```
CyclingIsOccurringException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

Description

Cycling is occurring.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

DeleteObservationsException Class

Summary

The number of observations to be deleted (set by `setObservationMax`) has grown too large.

```
public class Imsl.Stat.DeleteObservationsException :  IMSLException :
ISerializable
```

Constructors

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.

DeleteObservationsException

```
public DeleteObservationsException()
```

Description

The number of observations to be deleted (set by `setObservationMax`) has grown too large.

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.

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

```
DeleteObservationsException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The iteration did not converge.

```
public class Imsl.Stat.DidNotConvergeException : IMSLException :
ISerializable
```

Constructors

DidNotConvergeException

```
public DidNotConvergeException()
```

Description

The iteration did not converge.

DidNotConvergeException

```
public DidNotConvergeException(string message)
```

Description

The iteration did not converge.

Parameter

`message` – The error message that explains the reason for the exception.

DidNotConvergeException

```
public DidNotConvergeException(string s, System.Exception exception)
```

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

```
DidNotConvergeException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Different observations are being deleted from return matrix than were originally entered.

```
public class Imsl.Stat.DiffObsDeletedException : IMSLException :
ISerializable
```

Constructors

DiffObsDeletedException

```
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

```
DiffObsDeletedException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

EigenvalueException Class

Summary

An error occured in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check the input covariance matrix.

```
public class Imsl.Stat.EigenvalueException :  IMSLException :  ISerializable
```

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

```
EigenvalueException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

Description

Eigenvalue error.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

EmptyGroupException Class

Summary

There are no observations in a group. Cannot compute statistics.

`public class Imsl.Stat.EmptyGroupException : IMSLException : ISerializable`

Constructors

EmptyGroupException

`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

`public EmptyGroupException()`

Description

There are no observations in a group. Cannot compute statistics.

EmptyGroupException

`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

`public EmptyGroupException(string s, System.Exception exception)`

Description

There are no observations in a group. Cannot compute statistics.

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,
 System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The equality constraints and the bounds on the variables are found to be inconsistent.

```
public class Imsl.Stat.EqConstrInconsistentException :  IMSLException :
ISerializable
```

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

```
EqConstrInconsistentException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The problem is ill-conditioned.

```
public class Imsl.Stat.IllConditionedException : IMSLException :
ISerializable
```

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

```
IllConditionedException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The bound for the relative error is too small.

```
public class Imsl.Stat.IncreaseErrRelException :  IMSLException :
ISerializable
```

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

```
IncreaseErrRelException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

MatrixSingularException Class

Summary

The input matrix is singular.

```
public class Imsl.Stat.MatrixSingularException : IMSLException :
ISerializable
```

Constructors

MatrixSingularException

```
public MatrixSingularException()
```

Description

The input matrix is singular.

MatrixSingularException

```
public MatrixSingularException(string message)
```

Description

The input matrix is singular.

Parameter

`message` – The error message that explains the reason for the exception.

MatrixSingularException

```
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

```
MatrixSingularException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

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).

```
public class Imsl.Stat.MoreObsDelThanEnteredException :  IMSLException :
ISerializable
```

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

```
MoreObsDelThanEnteredException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

A negative frequency was encountered.

```
public class Imsl.Stat.NegativeFreqException : IMSLException : ISerializable
```

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` – AAn `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,
 System.Runtime.Serialization.StreamingContext context)
```

Description

A negative frequency was encountered.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

NegativeWeightException Class

Summary

A negative weight was encountered.

```
public class Imsl.Stat.NegativeWeightException : IMSLException :
ISerializable
```

Constructors

NegativeWeightException

```
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

```
public NegativeWeightException()
```

Description

A negative weight was encountered.

NegativeWeightException

```
public NegativeWeightException(string message)
```

Description

A negative weight was encountered.

Parameter

`message` – The error message that explains the reason for the exception.

NegativeWeightException

`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

`NegativeWeightException(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)`

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

Summary

The iteration has not made good progress.

```
public class Imsl.Stat.NewInitialGuessException :  IMSLException :
ISerializable
```

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

```
NewInitialGuessException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

NoConvergenceException Class

Summary

Convergence did not occur within the maximum number of iterations.

```
public class Imsl.Stat.NoConvergenceException : IMSLException : ISerializable
```

Constructors

NoConvergenceException

```
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

```
public NoConvergenceException()
```

Description

Convergence did not occur within the maximum number of iterations.

NoConvergenceException

```
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

```
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

```
NoConvergenceException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

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

Summary

No degrees of freedom error.

```
public class Imsl.Stat.NoDegreesOfFreedomException : IMSLException :
ISerializable
```

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.

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.

NoDegreesOfFreedomException

```
NoDegreesOfFreedomException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

NoVariationInputException Class

Summary

There is no variation in the input data.

```
public class Imsl.Stat.NoVariationInputException :  IMSLException :
ISerializable
```

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

```
NoVariationInputException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

No vector X satisfies all of the constraints.

```
public class Imsl.Stat.NoVectorXException : IMSLException : ISerializable
```

Constructors

NoVectorXException

```
public NoVectorXException()
```

Description

No vector X satisfies all of the constraints.

NoVectorXException

```
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

```
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

```
NoVectorXException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

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.

NonPosVarianceException Class

Summary

The problem is ill-conditioned.

```
public class Imsl.Stat.NonPosVarianceException : IMSLException :
ISerializable
```

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

```
NonPosVarianceException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

NonPosVarianceXYException Class

Summary

The problem is ill-conditioned.

```
public class Imsl.Stat.NonPosVarianceXYException : IMSLException :
ISerializable
```

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, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Maximum number of iterations exceeded.

```
public class Imsl.Stat.NonPositiveEigenvalueException :  IMSLException :
ISerializable
```

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()`

Description

Maximum number of iterations exceeded.

NonPositiveEigenvalueException

`public NonPositiveEigenvalueException(string message)`

Description

Maximum number of iterations exceeded.

Parameter

`message` – The error message that explains the reason for the exception.

NonPositiveEigenvalueException

`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.

NonPositiveEigenvalueException

```
NonPositiveEigenvalueException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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.

NoPositiveVarianceException Class

Summary

No variable has positive variance. The Mahalanobis distances cannot be computed.

```
public class Imsl.Stat.NoPositiveVarianceException :  IMSLException :
ISerializable
```

Constructors

NoPositiveVarianceException

```
public NoPositiveVarianceException()
```

Description

No variable has positive variance. The Mahalanobis distances cannot be computed.

NoPositiveVarianceException

```
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

```
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

`NoPositiveVarianceException(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)`

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

Summary

The function is not a Cumulative Distribution Function (CDF).

`public class Imsl.Stat.NotCDFException : IMSLException : ISerializable`

Constructors

NotCDFException

`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

`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

`public NotCDFException(double x1, double x2, double f1)`

Description

The function is not a Cumulative Distribution Function (CDF).

The CDF function is not monotone, F(x1) = F(x2). No unique inverse exists.

Parameters

`x1` – is the first point

`x2` – is the second point

`f1` – is the common value for F(x1) and F(x2)

NotCDFException

`public NotCDFException(double lowerBound, double upperBound, double xx, int i)`

Description

The function is not a Cumulative Distribution Function (CDF).

The cdf function is not a cumulative distribution function because its value at a cutpoint is out of the expected range, [plower,pupper].

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.

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

```
NotCDFException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Covariance matrix is not positive definite.

```
public class Imsl.Stat.NotPositiveDefiniteException : IMSLException :
ISerializable
```

Constructors

NotPositiveDefiniteException

```
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

```
public NotPositiveDefiniteException()
```

Description

Covariance matrix is not positive definite.

NotPositiveDefiniteException

```
public NotPositiveDefiniteException(string message)
```

Description

Covariance matrix is not positive definite.

Parameter

`message` – The error message that explains the reason for the exception.

NotPositiveDefiniteException

```
public NotPositiveDefiniteException(string s, System.Exception exception)
```

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
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Covariance matrix is not positive semi-definite.

```
public class Imsl.Stat.NotPositiveSemiDefiniteException : IMSLException :
ISerializable
```

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

```
NotPositiveSemiDefiniteException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Hessian matrix is not semi-definite.

```
public class Imsl.Stat.NotSemiDefiniteException : IMSLException :
ISerializable
```

Constructors

NotSemiDefiniteException

```
public NotSemiDefiniteException()
```

Description

Hessian matrix is not semi-definite.

NotSemiDefiniteException

```
public NotSemiDefiniteException(string message)
```

Description

Hessian matrix is not semi-definite.

Parameter

`message` – The error message that explains the reason for the exception.

NotSemiDefiniteException

```
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.

NotSemiDefiniteException

```
NotSemiDefiniteException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

No Variables can enter the model.

```
public class Imsl.Stat.NoVariablesEnteredException : IMSLException :
ISerializable
```

Constructors

NoVariablesEnteredException

```
public NoVariablesEnteredException()
```

Description

No Variables can enter the model.

NoVariablesEnteredException

```
public NoVariablesEnteredException(string message)
```

Description

No Variables can enter the model.

Parameter

`message` – The error message that explains the reason for the exception.

NoVariablesEnteredException

`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

`NoVariablesEnteredException(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)`

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

Summary

No variables can enter the model.

`public class Imsl.Stat.NoVariablesException : IMSLException : ISerializable`

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

```
NoVariablesException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

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

Summary

There is no variation in the input data.

```
public class Imsl.Stat.NoVariationInputException :  IMSLException :
ISerializable
```

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

```
NoVariationInputException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

No vector X satisfies all of the constraints.

`public class Imsl.Stat.NoVectorXException : IMSLException : ISerializable`

Constructors

NoVectorXException

`public NoVectorXException()`

Description

No vector X satisfies all of the constraints.

NoVectorXException

`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

`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

```
NoVectorXException(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The pooled variance-Covariance matrix is singular.

```
public class Imsl.Stat.PooledCovarianceSingularException : IMSLException :
ISerializable
```

Constructors

PooledCovarianceSingularException

```
public PooledCovarianceSingularException()
```

Description

The pooled variance-Covariance matrix is singular.

PooledCovarianceSingularException

```
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

```
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.

PooledCovarianceSingularException

```
PooledCovarianceSingularException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Rank of covariance matrix error.

```
public class Imsl.Stat.RankException : IMSLException : ISerializable
```

Constructors

RankException

```
public RankException(int rank, int nf)
```

Description

Rank of covariance matrix error.

Parameters

`rank` – A `int` which specifies the rank of the covariance matrix.

`nf` – A `int` which specifies the number of factors.

RankException

```
public RankException()
```

Description

Rank of covariance matrix error.

RankException

```
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

```
RankException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

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.

ScaleFactorZeroException Class

Summary

The computations cannot continue because a scale factor is zero.

```
public class Imsl.Stat.ScaleFactorZeroException : IMSLException :
ISerializable
```

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.

`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, System.Runtime.Serialization.StreamingContext context)
```

Description

The computations cannot continue because a scale factor is zero.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

SingularException Class

Summary

Covariance matrix is singular.

`public class Imsl.Stat.SingularException : IMSLException : ISerializable`

Constructors

SingularException

`public SingularException(int i)`

Description

Covariance matrix is singular.

Parameter

`i` – Variable i is linearly related to the other variables.

SingularException

`public SingularException()`

Description

Covariance matrix is singular.

SingularException

`public SingularException(string message)`

Description

Covariance matrix is singular.

Parameter

`message` – The error message that explains the reason for the exception.

SingularException

`public SingularException(string s, System.Exception exception)`

Description

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.

SingularException

```
SingularException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

Description

Covariance matrix is singular.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

SumOfWeightsNegException Class

Summary

The sum of the weights have become negative.

```
public class Imsl.Stat.SumOfWeightsNegException :  IMSLException :
ISerializable
```

Constructors

SumOfWeightsNegException

```
public SumOfWeightsNegException(int group)
```

Description

The sum of the weights have become negative.

Parameter

`group` – A `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.

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

```
SumOfWeightsNegException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

Description

The sum of the weights have become negative.

Parameters

`info` – The object that holds the serialized object data.

`context` – The contextual information about the source or destination.

TooManyCallsException Class

Summary

The number of calls to the function has exceeded the maximum number of iterations.

`public class Imsl.Stat.TooManyCallsException : IMSLException : ISerializable`

Constructors

TooManyCallsException

```
public TooManyCallsException()
```

Description

The number of calls to the function has exceeded the maximum number of iterations.

TooManyCallsException

```
public TooManyCallsException(string message)
```

Description

The number of calls to the function has exceeded the maximum number of iterations.

Parameter

`message` – The error message that explains the reason for the exception.

TooManyCallsException

```
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

```
TooManyCallsException(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Maximum number of function evaluations exceeded.

```
public class Imsl.Stat.TooManyFunctionEvaluationsException : IMSLException :
ISerializable
```

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.

Parameter

`message` – The error message that explains the reason for the exception.

TooManyFunctionEvaluationsException

```
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

```
TooManyFunctionEvaluationsException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

Maximum number of iterations exceeded.

```
public class Imsl.Stat.TooManyIterationsException :  IMSLException :
ISerializable
```

Constructors

TooManyIterationsException

```
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

```
TooManyIterationsException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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.

TooManyJacobianEvalException Class

Summary

Maximum number of Jacobian evaluations exceeded.

```
public class Imsl.Stat.TooManyJacobianEvalException :  IMSLException :
ISerializable
```

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.

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
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

More observations have been deleted than were originally entered (the sum of frequencies has become negative).

```
public class Imsl.Stat.TooManyObsDeletedException : IMSLException :
ISerializable
```

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

```
TooManyObsDeletedException(System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The variables are determined by the equality constraints.

```
public class Imsl.Stat.VarsDeterminedException : IMSLException :
ISerializable
```

Constructors

VarsDeterminedException

```
public VarsDeterminedException()
```

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

```
VarsDeterminedException(System.Runtime.Serialization.SerializationInfo
 info, System.Runtime.Serialization.StreamingContext context)
```

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

Summary

The computations cannot continue because the Euclidean norm of the column is equal to zero.

```
public class Imsl.Stat.ZeroNormException : IMSLException : ISerializable
```

Constructors

ZeroNormException

```
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

`public ZeroNormException()`

Description

The computations cannot continue because the Euclidean norm of the column is equal to zero.

ZeroNormException

`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

`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

```
ZeroNormException(System.Runtime.Serialization.SerializationInfo info,
 System.Runtime.Serialization.StreamingContext context)
```

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 27: References

References

Abe

Abe, S. (2001) *Pattern Classification: Neuro-Fuzzy Methods and their Comparison*, Springer-Verlag.

Abramowitz and Stegun

Abramowitz, Milton, and Irene A. Stegun (editors) (1964), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, National Bureau of Standards, Washington.

Afifi and Azen

Afifi, A.A. and S.P. Azen (1979), *Statistical Analysis: A Computer Oriented Approach*, 2d ed., Academic Press, New York.

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

Ahrens, J.H., and U. Dieter (1974), Computer methods for sampling from gamma, beta, Poisson, and binomial distributions, *Computing*, **12**, 223-246.

Akaike

Akaike, H., (1978), *Covariance Matrix Computation of the State Variable of a Stationary Gaussian Process*, Ann. Inst. Statist. Math. 30 , Part B, 499-504.

Akaike et al

Akaike, H. , Kitagawa, G., Arahata, E., Tada, F., (1979), Computer Science Monographs No. 13, The Institute of Statistical Mathematics, Tokyo.

Akima

Akima, H. (1970), A new method of interpolation and smooth curve fitting based on local procedures, *Journal of the ACM*, **17**, 589-602.

Akima, H. (1978), A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points, *ACM Transactions on Mathematical Software*, **4**, 148-159.

Anderberg

Anderberg, Michael R. (1973), *Cluster Analysis for Applications*, Academic Press, New York.

Anderson

Anderson, T.W. (1971), *The Statistical Analysis of Time Series*, John Wiley & Sons, New York.
Anderson, T. W. (1994) *The Statistical Analysis of Time Series*, John Wiley & Sons, New York.

Anderson and Bancroft

Anderson, R.L. and T.A. Bancroft (1952), *Statistical Theory in Research*, McGraw-Hill Book Company, New York.

Ashcraft

Ashcraft, C. (1987), *A vector implementation of the multifrontal method for large sparse symmetric positive definite systems*, Technical Report ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, Washington.

Ashcraft et al.

Ashcraft, C., R. Grimes, J. Lewis, B. Peyton, and H. Simon (1987), Progress in sparse matrix methods for large linear systems on vector supercomputers. *Intern. J. Supercomputer Applic.* , **1(4)**, 10-29.

Atkinson (1979)

Atkinson, A.C. (1979), A family of switching algorithms for the computer generation of beta random variates, *Biometrika*, **66**, 141-145.

Atkinson (1978)

Atkinson, Ken (1978), *An Introduction to Numerical Analysis*, John Wiley & Sons, New York.

Barrodale and Roberts

Barrodale, I., and F.D.K. Roberts (1973), An improved algorithm for discrete L1 approximation, *SIAM Journal on Numerical Analysis*, **10**, 839 848.

Barrodale, I., and F.D.K. Roberts (1974), Solution of an overdetermined system of equations in the l1 norm, *Communications of the ACM*, **17**, 319 320.

Barrodale, I., and C. Phillips (1975), Algorithm 495. Solution of an overdetermined system of linear equations in the Chebyshev norm, *ACM Transactions on Mathematical Software*, **1**, 264 270.

Bartlett, M. S.

Bartlett, M.S. (1935), Contingency table interactions, *Journal of the Royal Statistics Society Supplement*, 2, 248 252.

Bartlett, M. S. (1937) Some examples of statistical methods of research in agriculture and applied biology, *Supplement to the Journal of the Royal Statistical Society*, 4, 137-183.

Bartlett, M. (1937), The statistical conception of mental factors, *British Journal of Psychology*, 28, 97-104.

Bartlett, M.S. (1946), On the theoretical specification and sampling properties of autocorrelated time series, *Supplement to the Journal of the Royal Statistical Society*, 8, 27-41.

Bartlett, M.S. (1978), *Stochastic Processes*, 3rd. ed., Cambridge University Press, Cambridge.

Barnett

Barnett, A.R. (1981), An algorithm for regular and irregular Coulomb and Bessel functions of real order to machine accuracy, *Computer Physics Communication*, **21**, 297-314.

Barrett and Heal

Barrett, J.C., and M. J.R. Healy (1978), A remark on Algorithm AS 6: Triangular decomposition of a symmetric matrix, *Applied Statistics*, **27**, 379-380.

Bays and Durham

Bays, Carter, and S.D. Durham (1976), Improving a poor random number generator, *ACM Transactions on Mathematical Software*, **2**, 59-64.

Bendel and Mickey

Bendel, Robert B., and M. Ray Mickey (1978), Population correlation matrices for sampling experiments, *Communications in Statistics*, B7, 163 182.

Berry and Linoff

Berry, M. J. A. and Linoff, G. (1997) *Data Mining Techniques*, John Wiley & Sons, Inc.

Best and Fisher

Best, D.J., and N.I. Fisher (1979), Efficient simulation of the von Mises distribution, *Applied Statistics*, 28, 152 157.

Bishop

Bishop, C. M. (1995) *Neural Networks for Pattern Recognition*, Oxford University Press.

Bishop et al

Bishop, Yvonne M.M., Stephen E. Feinberg, and Paul W. Holland (1975), *Discrete Multivariate Analysis: Theory and Practice*, MIT Press, Cambridge, Mass.

Bjorck and Golub

Bjorck, Ake, and Gene H. Golub (1973), Numerical Methods for Computing Angles Between Subspaces, *Mathematics of Computation,*, **27**, 579 594.

Blom

Blom, Gunnar (1958), *Statistical Estimates and Transformed Beta-Variables*, John Wiley & Sons, New York.

Blom and Zegeling

Blom, JG, and Zegeling, PA (1994), A Moving-grid Interface for Systems of One-dimensional Time-dependent Partial Differential Equations, *ACM Transactions on Mathematical Software*, Vol 20, No.2, 194-214.

Boisvert

Boisvert, Ronald (1984), A fourth order accurate fast direct method of the Helmholtz equation, *Elliptic Problem solvers II*, (edited by G. Birkhoff and A. Schoenstadt), Academic Press, Orlando, Florida, 35-44.

Bosten and Battiste

Bosten, Nancy E., and E.L. Battiste (1974), Incomplete beta ratio, *Communications of the ACM*, **17**, 156-157.

Box and Jenkins

Box, G. E. P. and Jenkins, G. M. (1970) *Time Series Analysis: Forecasting and Control*, Holden-Day, Inc.

Box and Pierce

Box, G.E.P., and David A. Pierce (1970), Distribution of residual autocorrelations in autoregressive-integrated moving average time series models, *Journal of the American Statistical Association*, 65, 1509-1526.

Boyette

Boyette, James M. (1979), Random RC tables with given row and column totals, Applied Statistics, 28, 329 332.

Bradley

Bradley, J.V. (1968), Distribution-Free Statistical Tests, Prentice-Hall, New Jersey.

Breiman et al.

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees*, Chapman & Hall.

Brenan, Campbell, and Petzold

Brenan, K.E., S.L. Campbell, L.R. Petzold (1989), *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Elseview Science Publ. Co.

Brent

Brent, Richard P. (1973), *Algorithms for Minimization without Derivatives*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Breslow

Breslow, N.E. (1974), Covariance analysis of censored survival data, *Biometrics*, 30, 89 99.

Bridle

Bridle, J. S. (1990) *Probabilistic Interpretation of Feedforward Classification Network Outputs,* *with Relationships to Statistical Pattern Recognition*, in F. Fogelman Soulie and J. Herault (Eds.), *Neuralcomputing: Algorithms, Architectures and Applications*, Springer-Verlag, 227-236.

Brighamv

Brigham, E. Oran (1974), *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, New Jersey.

Brown

Brown, Morton E. (1983), MCDP4F, two-way and multiway frequency tables-measures of association and the log-linear model (complete and incomplete tables), *in BMDP Statistical Software, 1983 Printing with Additions*, (edited by W.J. Dixon), University of California Press, Berkeley.

Brown and Benedetti

Brown, Morton B. and Jacqualine K. Benedetti (1977), Sampling behavior and tests for correlation in two-way contingency tables, *Journal of the American Statistical Association,*, 42, 309 315.

Burgoyne

Burgoyne, F.D. (1963), Approximations to Kelvin functions, *Mathematics of Computation*, **83**, 295-298.

Calvo

Calvo, R. A. (2001) *Classifying Financial News with Neural Networks*, Proceedings of the 6th Australasian Document Computing Symposium.

Carlson

Carlson, B.C. (1979), Computing elliptic integrals by duplication, *Numerische Mathematik*, **33**, 1-16.

Carlson and Notis

Carlson, B.C., and E.M. Notis (1981), Algorithms for incomplete elliptic integrals, *ACM Transactions on Mathematical Software*, **7**, 398-403.

Carlson and Foley

Carlson, R.E., and T.A. Foley (1991),The parameter R^2 in multiquadric interpolation, *Computer Mathematical Applications*, **21**, 29-42.

Chen and Liu

Chen, C. and Liu, L., Joint Estimation of Model Parameters and Outlier Effects in Time Series, *Journal of the American Statistical Association*, Vol. 88, No.421, March 1993.

Cheng

Cheng, R.C.H. (1978), Generating beta variates with nonintegral shape parameters, *Communications of the ACM*, **21**, 317-322.

Clarkson and Jenrich

Clarkson, Douglas B. and Robert B Jenrich (1991), Computing extended maximum likelihood estimates for linear parameter models, submitted to *Journal of the Royal Statistical Society*, Series B, **53**, 417-426.

Cohen and Taylor

Cohen, E. Richard, and Barry N. Taylor (1986), *The 1986 Adjustment of the Fundamental Physical Constants*, Codata Bulletin, Pergamon Press, New York.

Cooley and Tukey

Cooley, J.W., and J.W. Tukey (1965), An algorithm for the machine computation of complex Fourier series, *Mathematics of Computation*, **19**, 297-301.

Cooper

Cooper, B.E. (1968), Algorithm AS4, An auxiliary function for distribution integrals, *Applied Statistics*, **17**, 190-192.

Cook and Weisberg

Cook, R. Dennis and Sanford Weisberg (1982), *Residuals and Influence in Regression*, Chapman and Hall, New York.

Courant and Hilbert

Courant, R., and D. Hilbert (1962), *Methods of Mathematical Physics*, Volume II, John Wiley & Sons, New York, NY.

Craven and Wahba

Craven, Peter, and Grace Wahba (1979), Smoothing noisy data with spline functions, *Numerische Mathematik*, **31**, 377-403.

Crowe et al.

Crowe, Keith, Yuan-An Fan, Jing Li, Dale Neaderhouser, and Phil Smith (1990), *A direct sparse linear equation solver using linked list storage*, IMSL Technical Report 9006, IMSL, Houston.

Davis and Rabinowitz

Davis, Philip F., and Philip Rabinowitz (1984), *Methods of Numerical Integration*, Academic Press, Orlando, Florida.

de Boor

de Boor, Carl (1978), *A Practical Guide to Splines*, Springer-Verlag, New York.

Dennis and Schnabel

Dennis, J.E., Jr., and Robert B. Schnabel (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, New Jersey.

Dongarra et al.

Dongarra, J.J., J.R. Bunch, C.B. Moler, and G.W. Stewart (1979), *LINPACK User's Guide*, SIAM, Philadelphia.

Draper and Smith

Draper, N.R., and H. Smith (1981), *Applied Regression Analysis*, 2nd. ed., John Wiley & Sons, New York.

DuCroz et al.

Du Croz, Jeremy, P. Mayes, and G. Radicati (1990), Factorization of band matrices using Level-3 BLAS, *Proceedings of CONPAR 90-VAPP IV*, Lecture Notes in Computer Science, Springer, Berlin, 222.

Duff et al.

Duff, I. S., A. M. Erisman, and J. K. Reid (1986), *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford.

Duff and Reid

Duff, I.S., and J.K. Reid (1983), The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, **9**, 302-325.

Duff, I.S., and J.K. Reid (1984), The multifrontal solution of unsymmetric sets of linear equations. *SIAM Journal on Scientific and Statistical Computing*, **5**, 633-641.

Elman

Elman, J. L. (1990) *Finding Structure in Time, Cognitive Science*, **14**, 179-211.

Enright and Pryce

Enright, W.H., and J.D. Pryce (1987), Two FORTRAN packages for assessing initial value methods, *ACM Transactions on Mathematical Software*, **13**, 1-22.

Farebrother and Berry

Farebrother, R.W., and G. Berry (1974), A remark on Algorithm AS 6: Triangular decomposition of a symmetric matrix, *Applied Statistics*, **23**, 477.

Fisher

Fisher, R.A. (1936), The use of multiple measurements in taxonomic problems, *Annals of Eugenics*, **7**, 179-188.

Fishman and Moore

Fishman, George S. and Louis R. Moore (1982), A statistical evaluation of multiplicative congruential random number generators with modulus 231 - 1, *Journal of the American Statistical Association*, **77**, 129-136.

Forsythe

Forsythe, G.E. (1957), Generation and use of orthogonal polynomials for fitting data with a digital computer, *SIAM Journal on Applied Mathematics*, **5**, 74-88.

Franke

Franke, R. (1982), Scattered data interpolation: Tests of some methods, *Mathematics of*

Computation, **38**, 181-200.

Furnival and Wilson

Furnival, G.M. and R.W. Wilson, Jr. (1974), Regressions by leaps and bounds, *Technometrics*, **16**, 499-511.

Garbow et al.

Garbow, B.S., J.M. Boyle, K.J. Dongarra, and C.B. Moler (1977), *Matrix Eigensystem Routines - EISPACK Guide Extension*, Springer-Verlag, New York.

Garbow, B.S., G. Giunta, J.N. Lyness, and A. Murli (1988), Software for an implementation of Weeks' method for the inverse Laplace transform problem, *ACM Transactions on Mathematical Software*, **14**, 163-170.

Gautschi

Gautschi, Walter (1968), Construction of Gauss-Christoffel quadrature formulas, *Mathematics of Computation*, **22**, 251-270.

Gear

Gear, C.W. (1971), Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, New Jersey.

Gear and Petzold

Gear, C.W. and Petzold, Linda R. (1984), ODE methods for the solution of differential/algebraic equations. *SIAM Journal of Numerical Analysis*, **21**, #4, 716.

Gentleman

Gentleman, W. Morven (1974), Basic procedures for large, sparse or weighted linear least squares problems, *Applied Statistics*, **23**, 448-454.

George and Liu

George, A., and J.W.H. Liu (1981), *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, New Jersey.

Gill and Murray

Gill, Philip E., and Walter Murray (1976), *Minimization subject to bounds on the variables*, NPL Report NAC 92, National Physical Laboratory, England.

Gill et al.

Gill, P.E., W. Murray, M.A. Saunders, and M.H. Wright (1985), Model building and practical aspects of nonlinear programming, in *Computational Mathematical Programming*, (edited by K. Schittkowski), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany.

Giudici

Giudici, P. (2003) *Applied Data Mining: Statistical Methods for Business and Industry*, John Wiley & Sons, Inc.

Goldfarb and Idnani

Goldfarb, D., and A. Idnani (1983), A numerically stable dual method for solving strictly convex quadratic programs, *Mathematical Programming*, **27**, 1-33.

Golub

Golub, G.H. (1973), Some modified matrix eigenvalue problems, *SIAM Review*, **15**, 318-334.

Golub and Van Loan

Golub, G.H., and C.F. Van Loan (1989), *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, Maryland.

Golub, Gene H., and Charles F. Van Loan (1983), *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland.

Golub and Welsch

Golub, G.H., and J.H. Welsch (1969), Calculation of Gaussian quadrature rules, *Mathematics of Computation*, **23**, 221-230.

Gregory and Karney

Gregory, Robert, and David Karney (1969), *A Collection of Matrices for Testing Computational Algorithms*, Wiley-Interscience, John Wiley & Sons, New York.

Griffin and Redfish

Griffin, R., and K A. Redish (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **13**, 54.

Grosse

Grosse, Eric (1980), Tensor spline approximation, *Linear Algebra and its Applications*, **34**, 29-41.

Guerra and Tapia

Guerra, V., and R. A. Tapia (1974), *A local procedure for error detection and data smoothing*, MRC Technical Summary Report 1452, Mathematics Research Center, University of Wisconsin, Madison.

Hageman and Young

Hageman, Louis A., and David M. Young (1981), *Applied Iterative Methods*, Academic Press, New York.

Hanson

Hanson, Richard J. (1986), Least squares with bounds and linear constraints, *SIAM Journal Sci. Stat. Computing*, 7, #3.

Hardy

Hardy, R.L. (1971), Multiquadric equations of topography and other irregular surfaces, *Journal of Geophysical Research*, **76**, 1905-1915.

Harman

Harman, Harry H. (1976), *Modern Factor Analysis*, 3d ed. revised, University of Chicago Press, Chicago.

Hart et al.

Hart, John F., E.W. Cheney, Charles L. Lawson, Hans J.Maehly, Charles K. Mesztenyi, John R. Rice, Henry G. Thacher, Jr., and Christoph Witzgall (1968), *Computer Approximations*, John Wiley & Sons, New York.

Healy

Healy, M.J.R. (1968), Algorithm AS 6: Triangular decomposition of a symmetric matrix, *Applied Statistics*, **17**, 195-197.

Hebb

Hebb, D. O. (1949) *The Organization of Behaviour: A Neuropsychological Theory*, John Wiley.

Herraman

Herraman, C. (1968), Sums of squares and products matrix, *Applied Statistics*, **17**, 289-292.

Higham

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.

Hill

Hill, G.W. (1970), Student's t-distribution, *Communications of the ACM*, **13**, 617-619.

Hindmarsh

Hindmarsh, A.C. (1974), *GEAR:* Ordinary Differential Equation System Solver, Lawrence Livermore National Laboratory Report UCID-30001, Revision 3, Lawrence Livermore National Laboratory, Livermore, Calif.

Hinkley

Hinkley, David (1977), On quick choice of power transformation, *Applied Statistics*, **26**, 67-69.

Hocking

Hocking, R.R. (1972), Criteria for selection of a subset regression: Which one should be used?, *Technometrics*, **14**, 967-970.

Hocking, R.R. (1973), A discussion of the two-way mixed model, *The American Statistician*, **27**, 148-152.

Hopfield

Hopfield, J. J. (1987) *Learning Algorithms and Probability Distributions in Feed-Forward and Feed-Back Networks*, Proceedings of the National Academy of Sciences, **84**, 8429-8433.

Huber

Huber, Peter J. (1981), *Robust Statistics*, John Wiley & Sons, New York.

Hutchinson

Hutchinson, J. M. (1994) *A Radial Basis Function Approach to Financial Timer Series Analysis*, Ph.D. dissertation, Massachusetts Institute of Technology.

Hull et al.

Hull, T.E., W.H. Enright, and K.R. Jackson (1976), *User's guide for DVERK–A subroutine for solving non-stiff ODEs*, Department of Computer Science Technical Report 100, University of Toronto.

Hwang and Ding

Hwang, J. T. G. and Ding, A. A. (1997) *Prediction Intervals for Artificial Neural Networks*, Journal of the American Statistical Society, **92**(438) 748-757.

Irvine et al.

Irvine, Larry D., Samuel P. Marin, and Philip W. Smith (1986), Constrained interpolation and smoothing, *Constructive Approximation*, **2**, 129-151.

Jackson et al.

Jackson, K.R., W.H. Enright, and T.E. Hull (1978), A theoretical criterion for comparing Runge-Kutta formulas, *SIAM Journal of Numerical Analysis*, **15**, 618-641.

Jacobs et al.

Jacobs, R. A., Jorday, M. I., Nowlan, S. J., and Hinton, G. E. (1991) Adaptive Mixtures of Local Experts, *Neural Computation*, **3(1)**, 79-87.

Jenkins

Jenkins, M.A. (1975), Algorithm 493: Zeros of a real polynomial, *ACM Transactions on Mathematical Software*, **1**, 178-189.

Jenkins and Traub

Jenkins, M.A., and J.F. Traub (1970), A three-stage algorithm for real polynomials using quadratic iteration, *SIAM Journal on Numerical Analysis*, 7, 545-566.

Jenkins, M.A., and J.F. Traub (1970), A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration, *Numerishe Mathematik*, **14**, 252-263.

Jenkins, M.A., and J.F. Traub (1972), Zeros of a complex polynomial, *Communications of the AC*M, **15**, 97- 99.

Jöhnk

Jöhnk, M.D. (1964), Erzeugung von Betaverteilten und Gammaverteilten Zufalls-zahlen, *Metrika*, **8**, 5-15.

Johnson and Kotz

Johnson, Norman L., and Samuel Kotz (1969), *Discrete Distributions*, Houghton Mifflin

Company, Boston.

Johnson, Norman L., and Samuel Kotz (1970a), *Continuous Univariate Distributions-1*, John Wiley & Sons, New York.

Johnson, Norman L., and Samuel Kotz (1970b), *Continuous Univariate Distributions-2*, John Wiley & Sons, New York.

Jöreskog

Jöreskog, M.D. (1977), Factor analysis by least squares and maximum-likelihood methods, *Statistical Methods for Digital Computers*, (edited by Kurt Enslein, Anthony Ralston, and Herbert S. Wilf), John Wiley & Sons, New York, 125-153.

Kaiser

Kaiser, H.F. (1963), Image analysis, *Problems in Measuring Change*, (edited by C. Harris), University of Wisconsin Press, Madison, Wis.

Kaiser and Caffrey

Kaiser, H.F. and J. Caffrey (1965), Alpha factor analysis, *Psychometrika*, **30**, 1-14.

Kachitvichyanukul

Kachitvichyanukul, Voratas (1982), *Computer generation of Poisson, binomial, and hypergeometric random variates*, Ph.D. dissertation, Purdue University, West Lafayette, Indiana.

Kendall and Stuart

Kendall, Maurice G., and Alan Stuart (1973), *The Advanced Theory of Statistics*, Volume II, *Inference and Relationship*, Third Edition, Charles Griffin & Company, London, Chapter 30.

Kennedy and Gentle

Kennedy, William J., Jr., and James E. Gentle (1980), *Statistical Computing*, Marcel Dekker, New York.

Kernighan and Richtie

Kernighan, Brian W., and Richtie, Dennis M. 1988, "The C Programming Language" Second Edition, **241**.

Kinnucan and Kuki

Kinnucan, P., and Kuki, H., (1968), *A single precision inverse error function subroutine*, Computation Center, University of Chicago.

Kirk

Kirk, Roger, E., (1982), "Experimental Design" Second Edition, *Procedures in Behavioral Sciences*, Brooks/Cole Publishing Company, Monterey, CA.

Kohonen

Kohonen, T. (1995) *Self-Organizing Maps*, Springer-Verlag.

Knuth

Knuth, Donald E. (1981), The Art of Computer Programming, Volume II: *Seminumerical Algorithms*, 2nd. ed., Addison-Wesley, Reading, Mass.

Lachenbruch

Lachenbruch, Peter A. (1975), *Discriminant Analysis*, Hafner Press, London.

Lawrence et al

Lawrence, S., Giles, C. L, Tsoi, A. C., Back, A. D. (1997) Face Recognition: A Convolutional Neural Network Approach, *IEEE Transactions on Neural Networks, Special Issue on Neural Networks and Pattern Recognition*, 8(1), 98-113.

Learmonth and Lewis

Learmonth, G.P., and P.A.W. Lewis (1973), *Naval Postgraduate School Random Number Generator Package LLRANDOM, NPS55LW73061A*, Naval Postgraduate School, Monterey, California.

Lehmann

Lehmann, E.L. (1975), *Nonparametrics: Statistical Methods Based on Ranks*, Holden-Day, San Francisco.

Levenberg

Levenberg, K. (1944), A method for the solution of certain problems in least squares, *Quarterly of Applied Mathematics*, **2**, 164-168.

Leavenworth

Leavenworth, B. (1960), Algorithm 25: Real zeros of an arbitrary function, *Communications of the ACM*, **3**, 602.

Lentini and Pereyra Pereyra, Victor (1978), PASVA3: An adaptive finite-difference FORTRAN program for first order nonlinear boundary value problems, in *Lecture Notes in Computer Science*, **76**, Springer-Verlag, Berlin, 67 88.

Lewis et al.

Lewis, P.A.W., A.S. Goodman, and J.M. Miller (1969), A pseudorandom number generator for the System/ 360, *IBM Systems Journal*, **8**, 136-146.

Li

Li, L. K. (1992) *Approximation Theory and Recurrent Networks*, Proc. Int. Joint Conference On Neural Networks, vol. II, 266-271.

Liepman

Liepman, David S. (1964), Mathematical constants, in *Handbook of Mathematical Functions*, Dover Publications, New York.

Lippmann

Lippmann, R. P. (1989) *Review of Neural Networks for Speech Recognition,*, Neural Computation, ,b¿I, 1-38.

Liu

Liu, J.W.H. (1987), *A collection of routines for an implementation of the multifrontal method*, Technical Report CS-87-10, Department of Computer Science, York University, North York, Ontario, Canada.

Liu, J.W.H. (1989), The multifrontal method and paging in sparse Cholesky factorization. *ACM Transactions on Mathematical Software*, **15**, 310-325.

Liu, J.W.H. (1990), *The multifrontal method for sparse matrix solution: theory and practice*, Technical Report CS-90-04, Department of Computer Science, York University, North York, Ontario, Canada.

Liu, J.W.H. (1986), On the storage requirement in the out-of-core multifrontal method for sparse factorization. *ACM Transactions on Mathematical Software*, **12**, 249-264.

Loh and Shih

Loh, W.-Y. and Shih, Y.-S. (1997) Split Selection Methods for Classification Trees, *Statistica Sinica*, **7**, 815-840.

Lyness and Giunta

Lyness, J.N. and G. Giunta (1986), A modification of the Weeks Method for numerical inversion of the Laplace transform, *Mathematics of Computation*, **47**, 313-322.

Madsen and Sincovec

Madsen, N.K., and R.F. Sincovec (1979), Algorithm 540: PDECOL, General collocation software for partial differential equations, *ACM Transactions on Mathematical Software*,**5**, #3, 326-351.

Maindonald

Maindonald, J.H. (1984), *Statistical Computation*, John Wiley & Sons, New York.

Mandic and Chambers

Mandic, D. P. and Chambers, J. A. (2001) *Recurrent Neural Networks for Prediction*, John Wiley & Sons, LTD.

Manning and Schütze

Manning, C. D. and Schütze, H. (1999) *Foundations of Statistical Natural Language Processing*, MIT Press.

Marquardt

Marquardt, D. (1963), An algorithm for least-squares estimation of nonlinear parameters, *SIAM Journal on Applied Mathematics*, **11**, 431-441.

Marsaglia

Marsaglia, G. (1972), The structure of linear congruential sequences, in *Applications of Number*

Theory to Numerical Analysis, (edited by S. K. Zaremba), Academic Press, New York, 249-286.

Martin and Wilkinson

Martin, R.S., and J.H. Wilkinson (1971), Reduction of the Symmetric Eigenproblem Ax = lBx and Related Problems to Standard Form, *Volume II, Linear Algebra Handbook*, Springer, New York.

Martin, R.S., and J.H. Wilkinson (1971), The Modified LR Algorithm for Complex Hessenberg Matrices, *Handbook, Volume II, Linear Algebra*, Springer, New York.

Mayle

Mayle, Jan, (1993), Fixed Income Securities Formulas for Price, Yield, and Accrued Interest, *SIA Standard Securities Calculation Methods*, Volume I, Third Edition, pages 17-35.

McCulloch and Pitts

McCulloch, W. S. and Pitts, W. (1943) A Logical Calculus for Ideas Imminent in Nervous Activity, *Bulletin of Mathematical Biophysics*, **5**, 115-133.

Michelli

Micchelli, C.A. (1986), Interpolation of scattered data: Distance matrices and conditionally positive definite functions, *Constructive Approximation*, **2**, 11-22.

Michelli et al.

Micchelli, C.A., T.J. Rivlin, and S. Winograd (1976), The optimal recovery of smooth functions, *Numerische Mathematik*, **26**, 279-285.

Micchelli, C.A., Philip W. Smith, John Swetits, and Joseph D. Ward (1985), Constrained L_p approximation, *Constructive Approximation*, **1**, 93-102.

Microsoft Excel User Education Team

Microsoft Excel 5 - Worksheet Function Reference, (1994), *Covers Microsoft Excel 5 for Windows*^{tm} and the *Apple Macintosh*^{tm}, Microsoft Press. Redmond, VA.

Moler and Stewart

Moler, C., and G.W. Stewart (1973), An algorithm for generalized matrix eigenvalue problems, *SIAM Journal on Numerical Analysis*, **10**, 241-256. *Covers Microsoft Excel 5 for Windows*^{tm}.

Moré et al.

Moré, Jorge, Burton Garbow, and Kenneth Hillstrom (1980), *User Guide for MINPACK-1*, Argonne National Laboratory Report ANL-80-74, Argonne, Illinois.

Müller

Müller, D.E. (1956), A method for solving algebraic equations using an automatic computer, *Mathematical Tables and Aids to Computation*, **10**, 208-215.

Murtagh

Murtagh, Bruce A. (1981), Advanced Linear Programming: Computation and Practice, McGraw-Hill, New York.

Murty

Murty, Katta G. (1983), *Linear Programming*, John Wiley and Sons, New York.

Neter and Wasserman

Neter, John, and William Wasserman (1974), *Applied Linear Statistical Models*, Richard D. Irwin, Homewood, Illinois.

Neter et al.

Neter, John, William Wasserman, and Michael H. Kutner (1983), *Applied Linear Regression Models*, Richard D. Irwin, Homewood, Illinois.

Østerby and Zlatev

Østerby, Ole, and Zahari Zlatev (1982), Direct Methods for Sparse Matrices, *Lecture Notes in Computer Science*, **157**, Springer-Verlag, New York.

Owen

Owen, D.B. (1962), *Handbook of Statistical Tables*, Addison-Wesley Publishing Company, Reading, Mass.

Owen, D.B. (1965), A special case of the bivariate non-central t distribution, *Biometrika*, **52**, 437-446.

Pao

Pao, Y. (1989) *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley Publishing.

Parlett

Parlett, B.N. (1980), *The Symmetric Eigenvalue Problem*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Pennington and Berzins

Pennington, S. V., Berzins, M., (1994), Software for First-order Partial Differential Equations. 63-99.

Petro

Petro, R. (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **13**, 624.

Petzold

Petzold, L.R. (1982), A description of DASSL: A differential/ algebraic system solver, Proceedings of the IMACS World Congress, Montreal, Canada.

Piessens et al.

Piessens, R., E. deDoncker-Kapenga, C.W. Überhuber, and D.K. Kahaner (1983), *QUADPACK*, Springer-Verlag, New York.

Poli and Jones

Poli, I. and Jones, R. D. (1994) *A Neural Net Model for Prediction*, Journal of the American Statistical Society, 89(425) 117-121.

Powell

Powell, M.J.D. (1978), A fast algorithm for nonlinearly constrained optimization calculations, *Numerical Analysis Proceedings, Dundee 1977, Lecture Notes in Mathematics*, (edited by G. A. Watson), **630**, Springer-Verlag, Berlin, Germany, 144-157.

Powell, M.J.D. (1985), On the quadratic programming algorithm of Goldfarb and Idnani, *Mathematical Programming Study*, **25**, 46-61.

Powell, M.J.D. (1988), *A tolerant algorithm for linearly constrained optimizations calculations*, DAMTP Report NA17, University of Cambridge, England.

Powell, M.J.D. (1989), *TOLMIN: A Fortran package for linearly constrained optimizations calculations*, DAMTP Report NA2, University of Cambridge, England.

Powell, M.J.D. (1983), *ZQPCVX a FORTRAN subroutine for convex quadratic programming,* DAMTP Report 1983/NA17, University of Cambridge, Cambridge, England.

Pregibon

Pregibon, Daryl (1981), Logistic regression diagnostics, *The Annals of Statistics*, **9**, 705-724.

Quinlan

Quinlan, J. R. (1993), C4.5 *Programs for Machine Learning*, Morgan Kaufmann.

Reed and Marks

Reed, R. D. and Marks, R. J. II (1999) *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, The MIT Press, Cambridge, MA.

Reinsch

Reinsch, Christian H. (1967), Smoothing by spline functions, *Numerische Mathematik*, **10**, 177-183.

Rice

Rice, J.R. (1983), *Numerical Methods, Software, and Analysis*, McGraw-Hill, New Yor.

Ripley

Ripley, B. D. (1994) Neural Networks and Related Methods for Classification, *Journal of the Royal Statistical Society B*, **56(3)**, 409-456.

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*, Cambridge University Press.

Rosenblatt

Rosenblatt, F. (1958) The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,Psychol. Rev., **65**, 386-408.

Rumelhart et al

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986) Learning Representations by Back-Propagating Errors, *Nature*, **323**, 533-536.

Rumelhart, D. E. and McClelland, J. L. eds. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, **1**, 318-362, MIT Press.

Saad and Schultz

Saad, Y., and M. H. Schultz (1986), GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems, *SIAM Journal of Scientific and Statistical Computing*, **7**, 856-869.

Sallas and Lionti

Sallas, William M., and Abby M. Lionti (1988), Some useful computing formulas for the nonfull rank linear model with linear equality restrictions, IMSL Technical Report 8805, IMSL, Houston.

Savage

Savage, I. Richard (1956), Contributions to the theory of rank order statistics–the two-sample case, *Annals of Mathematical Statistics*, **27**, 590-615.

Schittkowski

Schittkowski, K. (1987), *More test examples for nonlinear programming codes*, Springer-Verlag, Berlin, **74**.

Schittkowski, K. (1986), NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems, (edited by Clyde L. Monma), *Annals of Operations Research*, **5**, 485-500.

Schittkowski, K. (1980), Nonlinear programming codes, *Lecture Notes in Economics and Mathematical Systems*, **183**, Springer-Verlag, Berlin, Germany.

Schittkowski, K. (1983), On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function, *Mathematik Operationsforschung und Statistik, Series Optimization*, **14**, 197-216.

Schmeiser

Schmeiser, Bruce (1983), Recent advances in generating observations from discrete random variates, in *Computer Science and Statistics: Proceedings of the Fifteenth Symposium on the Interface*, (edited by James E. Gentle), North-Holland Publishing Company, Amsterdam, 154-160.

Schmeiser and Babu

Schmeiser, Bruce W., and A.J.G. Babu (1980), Beta variate generation via exponential majorizing functions, *Operations Research*, **28**, 917-926.

Schmeiser and Kachitvichyanukul

Schmeiser, Bruce, and Voratas Kachitvichyanukul (1981), *Poisson Random Variate Generation*, Research Memorandum 81–4, School of Industrial Engineering, Purdue University, West

Lafayette, Indiana.

Schmeiser and Lal

Schmeiser, Bruce W., and Ram Lal (1980), Squeeze methods for generating gamma variates, *Journal of the American Statistical Association*, **75**, 679-682.

Seidler and Carmichael

Seidler, Lee J. and Carmichael, D.R., (editors) (1980), *Accountants' Handbook*, Volume I, Sixth Edition, The Ronald Press Company, New York.

Shampine

Shampine, L.F. (1975), Discrete least squares polynomial fits, *Communications of the ACM*, **18**, 179-180.

Shampine and Gear

Shampine, L.F. and C.W. Gear (1979), A user's view of solving stiff ordinary differential equations, *SIAM Review*, **21**, 1-17.

Sincovec and Madsen

Sincovec, R.F., and N.K. Madsen (1975), Software for nonlinear partial differential equations, *ACM Transactions on Mathematical Software*, **1**, #3, 232-260.

Singleton

Singleton, T.C. (1969), Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **12**, 185-187.

Smith et al.

Smith, B.T., J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler (1976), *Matrix Eigensystem Routines – EISPACK Guide*, Springer-Verlag, New York.

Smith

Smith, M. (1993) *Neural Networks for Statistical Modeling*, New York: Van Nostrand Reinhold.

Smith

Smith, P.W. (1990), On knots and nodes for spline interpolation, *Algorithms for Approximation II*, J.C. Mason and M.G. Cox, Eds., Chapman and Hall, New York.

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

Spellucci, P. (1998), A new technique for inconsistent problems in the SQP method. *Math. Meth. of Oper.* Res.,**47**, 355-500, Physica Verlag, Heidelberg, Germany.

Stewart

Stewart, G.W. (1973), Introduction to Matrix Computations, Academic Press, New York.

Stoer

Stoer, J. (1985), Principles of sequential quadratic programming methods for solving nonlinear programs, in *Computational Mathematical Programming*, (edited by K. Schittkowski), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany.

Strecok

Strecok, Anthony J. (1968), On the calculation of the inverse of the error function, *Mathematics of Computation*, **22**, 144-158.

Stroud and Secrest

Stroud, A.H., and D.H. Secrest (1963), *Gaussian Quadrature Formulae*, Prentice-Hall, Englewood Cliffs, New Jersey.

Studenmund

Studenmund, A. H. (1992) *Using Economics: A Practical Guide*, New York: Harper Collins.

Swingler

Swingler, K. (1996) *Applying Neural Networks: A Practical Guide*, Academic Press.

Temme

Temme, N.M (1975), On the numerical evaluation of the modified Bessel Function of the third kind, *Journal of Computational Physics*, **19**, 324-337.

Tesauro

Tesauro, G. (1990) Neurogammon Wins Computer Olympiad, *Neural Computation*, **1**, 321-323.

Tezuka

Tezuka, S. (1995), *Uniform Random Numbers: Theory and Practice.* Academic Publishers, Boston.

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

Tukey, John W. (1962), The future of data analysis, *Annals of Mathematical Statistics*, **33**, 1-67.

Velleman and Hoaglin

Velleman, Paul F., and David C. Hoaglin (1981), *Applications, Basics, and Computing of Exploratory Data Analysis*, Duxbury Press, Boston.

Verwer et al

Verwer, J. G., Blom, J. G., Furzeland, R. M., and Zegeling, P. A. (1989), A moving-grid method for one-dimensional PDEs Based on the Method of Lines, *Adaptive Methods for Partial Differential Equations*, Eds., J. E. Flaherty, P. J. Paslow, M. S. Shephard, and J. D. Vasiilakis, SIAM Publications, Philadelphia, PA (USA) pp. 160-175.

Walker

Walker, H.F. (1988), Implementation of the GMRES method using Householder transformations, *SIAM Journal of Scientific and Statistical Computing*, **9**, 152-163.

Warner and Misra

Warner, B. and Misra, M. (1996) Understanding Neural Networks as Statistical Tools, *The American Statistician*, **50(4)** 284-293.

Watkins

Watkins, David S., L. Elsner (1991), Convergence of algorithm of decomposition type for the eigenvalue problem, *Linear Algebra Applications*, **143**, pp. 29-47.

Weeks

Weeks, W.T. (1966), Numerical inversion of Laplace transforms using Laguerre functions, *J. ACM*, **13**, 419-429.

Werbos

Werbos, P. (1974) Beyond Regression: *New Tools for Prediction and Analysis in the Behavioral Science*, PhD thesis, Harvard University, Cambridge, MA.Werbos, P. (1990) Backpropagation Through Time: What It Does and How to do It, Proc.*IEEE*, **78**, 1550-1560.

Williams and Zipser

Williams, R. J. and Zipser, D. (1989) A Learning Algorithm for Continuously Running Fully Recurrent Neural Networks, *Neural Computation*, **1**, 270-280.

Wilmott et al

Wilmott, P., Howison, and S., Dewynne, J., (1996), *The Mathematics of Financial Derivatives (A Student Introduction)*, Cambridge Univ. Press, New York, NY. 317 pages.

Witten and Frank

Witten, I. H. and Frank, E. (2000) Data Mining: *Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann Publishers.

Wu

Wu, S-I (1995) Mirroring Our Thought Processes, *IEEE Potentials*, **14**, 36-41.

Index