SSJ User's Guide

Package `functions`

Functions utilities

Version: June 18, 2010

This package contains a few utilities classes representing univariate mathematical functions. They are useful, for example, when one wants to pass an arbitrary function of one variable as argument to a method. They allow one to apply mathematical operations like squaring, power, etc. on generic functions. There are also utilities for numerical differentiation and integration.

Contents

MathFunction

This interface should be implemented by classes which represent univariate mathematical functions. It is used to pass an arbitrary function of one variable as argument to another function. For example, it is used in `RootFinder` to find the zeros of a function.

```
package umontreal.iro.lecuyer.functions;

public interface MathFunction

   public double evaluate (double x);
```

Returns the value of the function evaluated at x.

MathFunctionUtil

Provides utility methods for computing derivatives and integrals of functions.

```
package umontreal.iro.lecuyer.functions;
```

```
public class MathFunctionUtil
```

```
   public static double H = 1e-6;
```

Step length in x to compute derivatives. Default: 10^{-6}.

```
   public static int NUMINTERVALS = 1024;
```

Default number of intervals for Simpson's integral.

```
   public static double derivative (MathFunction func, double x)
```

Returns the first derivative of the function `func` evaluated at `x`. If the given function implements `MathFunctionWithFirstDerivative`, this method calls `MathFunctionWithFirstDerivative.derivative (double)`. Otherwise, if the function implements `MathFunctionWithDerivative`, this method calls `MathFunctionWithDerivative.derivative (double, int)`. If the function does not implement any of these two interfaces, the method uses `finiteCenteredDifferenceDerivative (MathFunction, double, double)` to obtain an estimate of the derivative.

```
   public static double derivative (MathFunction func, double x, int n)
```

Returns the nth derivative of function `func` evaluated at `x`. If $n = 0$, this returns $f(x)$. If $n = 1$, this calls `derivative (MathFunction, double)` and returns the resulting first derivative. Otherwise, if the function implements `MathFunctionWithDerivative`, this method calls `MathFunctionWithDerivative.derivative (double, int)`. If the function does not implement this interface, the method uses `finiteCenteredDifferenceDerivative (MathFunction, double, int, double)` if n is even, or `finiteDifferenceDerivative (MathFunction, double, int, double)` if n is odd, to obtain a numerical approximation of the derivative.

```
   public static double finiteDifferenceDerivative (
               MathFunction func, double x, int n, double h)
```

Computes and returns an estimate of the nth derivative of the function $f(x)$. This method estimates

$$\frac{d^n f(x)}{dx^n},$$

the nth derivative of $f(x)$ evaluated at x. This method first computes $f_i = f(x + i\epsilon)$, for $i = 0, \ldots, n$, with $\epsilon = h^{1/n}$. The estimate is then given by $\Delta^n f_0/h$, where $\Delta^n f_i = \Delta^{n-1} f_{i+1} - \Delta^{n-1} f_i$, and $\Delta f_i = f_{i+1} - f_i$.

```
   public static double finiteCenteredDifferenceDerivative (
               MathFunction func, double x, double h)
```

Returns $(f(x+h) - f(x-h))/(2h)$, an estimate of the first derivative of $f(x)$ using centered differences.

`public static double finiteCenteredDifferenceDerivative ( MathFunction func, double x, int n, double h)`

Computes and returns an estimate of the nth derivative of the function $f(x)$ using finite centered differences. If n is even, this method returns `finiteDifferenceDerivative (func, x - ϵ*n/2, n, h)`, with $h = \epsilon^n$.

`public static double[][] removeNaNs (double[] x, double[] y)`

Removes any point `(NaN, y)` or `(x, NaN)` from `x` and `y`, and returns a 2D array containing the filtered points. This method filters each pair (`x[i]`, `y[i]`) containing at least one NaN element. It constructs a 2D array containing the two filtered arrays, whose size is smaller than or equal to `x.length`.

`public static double integral (MathFunction func, double a, double b)`

Returns the integral of the function `func` over $[a, b]$. If the given function implements `MathFunctionWithIntegral`, this returns `MathFunctionWithIntegral.integral (double, double)`. Otherwise, this calls `simpsonIntegral (MathFunction, double, double, int)` with `NUMINTERVALS` intervals.

`public static double simpsonIntegral (MathFunction func, double a, double b, int numIntervals)`

Computes and returns an approximation of the integral of `func` over $[a, b]$, using the Simpson's 1/3 method with `numIntervals` intervals. This method estimates

$$\int_a^b f(x)dx,$$

where $f(x)$ is the function defined by `func` evaluated at x, by dividing $[a, b]$ in $n =$ `numIntervals` intervals of length $h = (b - a)/n$. The integral is estimated by

$$\frac{h}{3}(f(a) + 4f(a + h) + 2f(a + 2h) + 4f(a + 3h) + \cdots + f(b))$$

This method assumes that $a \leq b < \infty$, and n is even.

`public static double gaussLobatto (MathFunction func, double a, double b, double tol)`

Computes and returns a numerical approximation of the integral of $f(x)$ over $[a, b]$, using Gauss-Lobatto adaptive quadrature with 5 nodes, with tolerance `tol`. This method estimates

$$\int_a^b f(x)dx,$$

where $f(x)$ is the function defined by `func`. Whenever the estimated error is larger than `tol`, the interval $[a, b]$ will be halved in two smaller intervals, and the method will recursively call itself on the two smaller intervals until the estimated error is smaller than `tol`.

`public static double gaussLobatto (MathFunction func, double a, double b, double tol, double[][] T)`

Similar to method `gaussLobatto(MathFunction, double, double, double)`, but also returns in `T[0]` the subintervals of integration, and in `T[1]`, the partial values of the integral

over the corresponding subintervals. Thus `T[0][0]` $= x_0 = a$ and `T[0][n]` $= x_n = b$; `T[1][i]` contains the value of the integral over the subinterval $[x_{i-1}, x_i]$; we also have `T[1][0]` $= 0$. The sum over all `T[1][i]`, for $i = 1, \ldots, n$ gives the value of the integral over $[a, b]$, which is the value returned by this method. *WARNING:* The user *must reserve* the 2 elements of the first dimension (`T[0]` and `T[1]`) before calling this method.

MathFunctionWithFirstDerivative

Represents a mathematical function whose derivative can be computed using `derivative`.

```
package umontreal.iro.lecuyer.functions;

public interface MathFunctionWithFirstDerivative extends MathFunction

   public double derivative (double x);
```

Computes (or estimates) the first derivative of the function at point `x`.

MathFunctionWithDerivative

Represents a mathematical function whose nth derivative can be computed using `derivative`.

```
package umontreal.iro.lecuyer.functions;

public interface MathFunctionWithDerivative extends MathFunction

   public double derivative (double x, int n);
```

Computes (or estimates) the nth derivative of the function at point `x`. For $n = 0$, this returns the result of `evaluate`.

MathFunctionWithIntegral

Represents a mathematical function whose integral can be computed by the `integral` method.

```
package umontreal.iro.lecuyer.functions;

public interface MathFunctionWithIntegral extends MathFunction

   public double integral (double a, double b);
```

Computes (or estimates) the integral of the function over the interval $[a, b]$.

PowerMathFunction

Represents a function computing $(af(x) + b)^p$ for a user-defined function $f(x)$ and power p.

```
package umontreal.iro.lecuyer.functions;

public class PowerMathFunction implements MathFunction
```

`public PowerMathFunction (MathFunction func, double power)`

Constructs a new power function for function `func` and power `power`. The values of the constants are $a = 1$ and $b = 0$.

`public PowerMathFunction (MathFunction func, double a, double b, double power)`

Constructs a new power function for function `func`, power `power`, and constants `a` and `b`.

`public MathFunction getFunction ()`

Returns the function $f(x)$.

`public double getA ()`

Returns the value of a.

`public double getB ()`

Returns the value of b.

`public double getPower ()`

Returns the power p.

ShiftedMathFunction

Represents a function computing $f(x) - \delta$ for a user-defined function $f(x)$ and shift δ.

```
package umontreal.iro.lecuyer.functions;

public class ShiftedMathFunction implements MathFunction
```

`public ShiftedMathFunction (MathFunction func, double delta)`

Constructs a new function shifting the function `func` by a shift `delta`.

`public MathFunction getFunction ()`

Returns the function $f(x)$.

`public double getDelta ()`

Returns the shift $\delta =$ `delta`.

SqrtMathFunction

Represents a function computing the square root of another function $f(x)$.

```
package umontreal.iro.lecuyer.functions;

public class SqrtMathFunction implements MathFunction
```

`public SqrtMathFunction (MathFunction func)`

Computes and returns the square root of the function `func`.

`public MathFunction getFunction()`

Returns the function associated with this object.

SquareMathFunction

Represents a function computing $(af(x) + b)^2$ for a user-defined function $f(x)$.

```
package umontreal.iro.lecuyer.functions;

public class SquareMathFunction implements MathFunctionWithFirstDerivative
```

`public SquareMathFunction (MathFunction func)`

Constructs a new square function for function `func`. The values of the constants are $a = 1$ and $b = 0$.

`public SquareMathFunction (MathFunction func, double a, double b)`

Constructs a new power function for function `func`, and constants `a` and `b`.

`public MathFunction getFunction()`

Returns the function $f(x)$.

`public double getA()`

Returns the value of a.

`public double getB()`

Returns the value of b.

AverageMathFunction

Represents a function computing the average of several functions. Let $f_0(x), \ldots, f_{n-1}(x)$ be a set of n functions. This function represents the average

$$f(x) = \frac{1}{n} \sum_{i=0}^{n-1} f_i(x).$$

```
package umontreal.iro.lecuyer.functions;

public class AverageMathFunction implements MathFunction
```

`public AverageMathFunction (MathFunction... func)`

Constructs a function computing the average of the functions in the array `func`.

`public MathFunction[] getFunctions()`

Returns the functions being averaged.

IdentityMathFunction

Represents the identity function $f(x) = x$.

```
package umontreal.iro.lecuyer.functions;

public class IdentityMathFunction implements MathFunction
```

PiecewiseConstantFunction

Represents a piecewise-constant function.

```
package umontreal.iro.lecuyer.functions;

public class PiecewiseConstantFunction implements MathFunction
```

`public PiecewiseConstantFunction (double[] x, double[] y)`

Constructs a new piecewise-constant function with X and Y coordinates given by `x` and `y`.

`public double[] getX()`

Returns the X coordinates of the function.

`public double[] getY()`

Returns the Y coordinates of the function.

Polynomial

Represents a polynomial of degree n in power form. Such a polynomial is of the form

$$p(x) = c_0 + c_1 x + \cdots + c_n x^n, \tag{1}$$

where $c_0, \ldots, c_n$ are the coefficients of the polynomial.

```
package umontreal.iro.lecuyer.functions;

public class Polynomial implements MathFunction
```

`public Polynomial (double... coeff)`

Constructs a new polynomial with coefficients `coeff`. The value of `coeff[i]` in this array corresponds to c_i.

`public int getDegree ()`

Returns the degree of this polynomial.

`public double[] getCoefficients ()`

Returns an array containing the coefficients of the polynomial.

`public double getCoefficient (int i)`

Returns the ith coefficient of the polynomial.

`public void setCoefficients (double... coeff)`

Sets the array of coefficients of this polynomial to `coeff`.

`public Polynomial derivativePolynomial (int n)`

Returns a polynomial corresponding to the nth derivative of this polynomial.

`public Polynomial integralPolynomial (double c)`

Returns a polynomial representing the integral of this polynomial. This integral is of the form

$$\int p(x)dx = c + c_0 x + \frac{c_1 x^2}{2} + \cdots + \frac{c_n x^{n+1}}{n+1},$$

where c is a user-defined constant.