SSJ User's Guide

Package `probdist`

Probability Distributions

Version: February 15, 2012

This package provides tools to compute densities, mass functions, distribution functions and their inverses, and reliability functions, for various continuous and discrete probability distributions. It also offers facilities for estimating the parameters of some distributions from a data set.

Contents

Overview . . . 2

General Classes **4**

Distribution . . . 4

DiscreteDistributionInt . . . 5

ContinuousDistribution . . . 7

DistributionFactory . . . 9

InverseDistFromDensity . . . 11

Discrete Distributions over Integers **13**

BernoulliDist . . . 13

BinomialDist . . . 15

GeometricDist . . . 18

HypergeometricDist . . . 20

LogarithmicDist . . . 22

NegativeBinomialDist . . . 24

PascalDist . . . 27

PoissonDist . . . 28

UniformIntDist . . . 30

ConstantIntDist . . . 32

Discrete Distributions over Real Numbers **33**

DiscreteDistribution . . . 33

ConstantDist . . . 35

EmpiricalDist . . . 36

Continuous Distributions **38**
BetaDist . . . 38
BetaSymmetricalDist . . . 41
CauchyDist . . . 43
ChiDist . . . 45
ChiSquareDist . . . 47
ChiSquareDistQuick . . . 49
ChiSquareNoncentralDist . . . 50
ErlangDist . . . 52
ExponentialDist . . . 54
ExponentialDistFromMean . . . 56
ExtremeValueDist . . . 57
FatigueLifeDist . . . 59
FisherFDist . . . 61
FoldedNormalDist . . . 63
FrechetDist . . . 65
GammaDist . . . 67
GammaDistFromMoments . . . 69
GumbelDist . . . 70
HalfNormalDist . . . 72
HyperbolicSecantDist . . . 74
HypoExponentialDist . . . 76
HypoExponentialDistQuick . . . 78
InverseGammaDist . . . 80
InverseGaussianDist . . . 82
JohnsonSBDist . . . 84
JohnsonSUDist . . . 86
LaplaceDist . . . 88
LogisticDist . . . 90
LoglogisticDist . . . 92
LognormalDist . . . 94
LognormalDistFromMoments . . . 96
NakagamiDist . . . 97

NormalDist 99
NormalDistQuick 101
NormalInverseGaussianDist 102
ParetoDist 104
Pearson5Dist 106
Pearson6Dist 108
PiecewiseLinearEmpiricalDist 110
PowerDist 112
RayleighDist 114
StudentDist 116
StudentDistQuick 118
TriangularDist 119
UniformDist 122
WeibullDist 124
TruncatedDist 126

Empirical Distribution Functions (EDF) 128

AndersonDarlingDist 128
AndersonDarlingDistQuick 129
CramerVonMisesDist 130
KolmogorovSmirnovPlusDist 132
KolmogorovSmirnovDist 134
KolmogorovSmirnovDistQuick 136
WatsonGDist 137
WatsonUDist 139

Overview

This package contains a set of Java classes providing methods to compute mass, density, distribution, complementary distribution, and inverse distribution functions for some discrete and continuous probability distributions. It also provides methods to estimate the parameters of some distributions from empirical data. It does not generate random variates; for that, see the package `randvar`. It is possible to plot the density or the cumulative probabilities of a distribution function either on screen, or in a LaTeX file, but for this, one has to use the package `charts`.

Distributions

We recall that the *distribution function* of a *continuous* random variable X with *density* f over the real line is

$$F(x) = P[X \le x] = \int_{-\infty}^{x} f(s)ds \tag{1}$$

while that of a *discrete* random variable X with *mass function* p over a fixed set of real numbers $x_0 < x_1 < x_2 < \cdots$ is

$$F(x) = P[X \le x] = \sum_{x_i \le x} p(x_i), \tag{2}$$

where $p(x_i) = P[X = x_i]$. For a discrete distribution over the set of integers, one has

$$F(x) = P[X \le x] = \sum_{s=-\infty}^{x} p(s), \tag{3}$$

where $p(s) = P[X = s]$.

We define $\bar{F}$, the *complementary distribution function* of X, by

$$\bar{F}(x) = P[X \ge x]. \tag{4}$$

With this definition of $\bar{F}$, one has $\bar{F}(x) = 1 - F(x)$ for continuous distributions and $\bar{F}(x) = 1 - F(x-1)$ for discrete distributions over the integers. This definition is *non-standard* for the discrete case: we have $\bar{F}(x) = P[X \ge x]$ instead of $\bar{F}(x) = P[X > x] = 1 - F(x)$. We find it more convenient especially for computing p-values in goodness-of-fit tests.

The *inverse distribution function* is defined as

$$F^{-1}(u) = \inf\{x \in \mathbb{R} : F(x) \ge u\}, \tag{5}$$

for $0 \le u \le 1$. This function F^{-1} is often used, among other things, to generate the random variable X by inversion, by passing a $U(0, 1)$ random variate as the value of u.

The package `probdist` offers two types of tools for computing p, f, F, $\bar{F}$, and F^{-1}: *static methods*, for which no object needs to be created, and methods associated with *distribution* *objects*. Standard distributions are implemented each in their own class. Constructing an object from one of these classes can be convenient if F, $\bar{F}$, etc., has to be evaluated several times for the same distribution. In certain cases (for the Poisson distribution, for example), creating the distribution object would precompute tables that would speed up significantly all subsequent method calls for computing F, $\bar{F}$, etc. This trades memory, plus a one-time setup cost, for speed. In addition to the non-static methods, the distribution classes also provide static methods that do not require the creation of an object.

The distribution classes extend one of the (abstract) classes `DiscreteDistribution` and `ContinuousDistribution` (which both implement the interface `Distribution`) for discrete and continuous distributions over the real numbers, or `DiscreteDistributionInt`, for discrete distributions over the non-negative integers.

For example, the class `PoissonDist` extends `DiscreteDistributionInt`. Calling a static method from this class will compute the corresponding probability from scratch. Constructing a `PoissonDist` object, on the other hand, will precompute tables that contain the probability terms and the distribution function for a given parameter λ (the mean of the Poisson distribution). These tables will then be used whenever a method is called for the corresponding object. This second approach is recommended if some of F, $\bar{F}$, etc., has to be computed several times for the same parameter λ. As a rule of thumb, creating objects and using their methods is faster than just using static methods as soon as two or three calls are made, unless the parameters are large.

In fact, only the non-negligible probability terms (those that exceed the threshold `DiscreteDistributionInt.EPSILON`) are stored in the tables. For F and $\bar{F}$, a single table actually contains $F(x)$ for $F(x) \le 1/2$ and $1 - F(x)$ for $F(x) > 1/2$. When the distribution parameters are so large that the tables would take too much space, these are not created and the methods automatically call their static equivalents instead of using tables.

Objects that implement the interface `Distribution` (and sometimes the abstract class `ContinuousDistribution`) are required by some methods in package `randvar` and also in classes `GofStat` and `GofFormat` of package `gof`.

Some of the classes also provide methods that compute parameter estimations of the corresponding distribution from a set of empirical observations, in most cases based on the maximum likelihood method.

Distribution

This interface should be implemented by all classes supporting discrete and continuous distributions. It specifies the signature of methods that compute the distribution function $F(x)$, the complementary distribution function $\bar{F}(x)$, and the inverse distribution function $F^{-1}(u)$. It also specifies the signature of methods that returns the mean, the variance and the standard deviation.

```
package umontreal.iro.lecuyer.probdist;


public interface Distribution
```

```
public double cdf (double x);
```

Returns the distribution function $F(x)$.

```
public double barF (double x);
```

Returns $\bar{F}(x) = 1 - F(x)$.

```
public double inverseF (double u);
```

Returns the inverse distribution function $F^{-1}(u)$, defined in (5).

```
public double getMean();
```

Returns the mean of the distribution function.

```
public double getVariance();
```

Returns the variance of the distribution function.

```
public double getStandardDeviation();
```

Returns the standard deviation of the distribution function.

```
public double[] getParams();
```

Returns the parameters of the distribution function in the same order as in the constructors.

DiscreteDistributionInt

Classes implementing discrete distributions over the integers should inherit from this class. It specifies the signatures of methods for computing the mass function (or probability) $p(x) = P[X = x]$, distribution function $F(x)$, complementary distribution function $\bar{F}(x)$, and inverse distribution function $F^{-1}(u)$, for a random variable X with a discrete distribution over the integers.

WARNING: the complementary distribution function is defined as $\bar{F}(j) = P[X \geq j]$ (for integers j, so that for discrete distributions in SSJ, $F(j) + \bar{F}(j) \neq 1$ since both include the term $P[X = j]$.

The implementing classes provide both static and non-static methods to compute the above functions. The non-static methods require the creation of an object of class `DiscreteDistributionInt`; all the non-negligible terms of the mass and distribution functions will be precomputed by the constructor and kept in arrays. Subsequent accesses will be very fast. The static methods do not require the construction of an object. These static methods are not specified in this abstract class because the number and types of their parameters depend on the distribution. When methods have to be called several times with the same parameters for the distributions, it is usually more efficient to create an object and use its non-static methods instead of the static ones. This trades memory for speed.

```
package umontreal.iro.lecuyer.probdist;

public abstract class DiscreteDistributionInt implements Distribution
```

```
public static double EPSILON = 1.0e-16;
```

Environment variable that determines what probability terms can be considered as negligible when building precomputed tables for distribution and mass functions. Probabilities smaller than `EPSILON` are not stored in the `DiscreteDistribution` objects (such as those of class `PoissonDist`, etc.), but are computed directly each time they are needed (which should be very seldom). The default value is set to 10^{-16}.

```
public abstract double prob (int x);
```

Returns $p(x)$, the probability of x.

```
public double cdf (double x)
```

Returns the distribution function F evaluated at x (see (2)). Calls the `cdf(int)` method.

```
public abstract double cdf (int x);
```

Returns the distribution function F evaluated at x (see (2)).

```
public double barF (double x)
```

Returns $\bar{F}(x)$, the complementary distribution function. Calls the `barF(int)` method.

`public double barF (int x)`

Returns $\bar{F}(x)$, the complementary distribution function. *See the WARNING above.*

`public int getXinf()`

Returns the lower limit x_a of the support of the probability mass function. The probability is 0 for all $x < x_a$.

`public int getXsup()`

Returns the upper limit x_b of the support of the probability mass function. The probability is 0 for all $x > x_b$.

`public double inverseF (double u)`

Returns the inverse distribution function $F^{-1}(u)$, where $0 \le u \le 1$. Calls the `inverseFInt` method.

`public int inverseFInt (double u)`

Returns the inverse distribution function $F^{-1}(u)$, where $0 \le u \le 1$. The default implementation uses binary search.

ContinuousDistribution

Classes implementing continuous distributions should inherit from this base class. Such distributions are characterized by a *density* function $f(x)$, thus the signature of a `density` method is supplied here. This class also provides default implementations for $\bar{F}(x)$ and for $F^{-1}(u)$, the latter using the Brent-Dekker method to find the inverse of a generic distribution function F.

```
package umontreal.iro.lecuyer.probdist;

public abstract class ContinuousDistribution implements Distribution
```

```
public abstract double density (double x);
```

Returns $f(x)$, the density evaluated at x.

```
public double barF (double x)
```

Returns the complementary distribution function. The default implementation computes $\bar{F}(x) = 1 - F(x)$.

```
public double inverseBrent (double a, double b, double u, double tol)
```

Computes the inverse distribution function $x = F^{-1}(u)$, using the Brent-Dekker method. The interval $[a, b]$ *must* contain the root x such that $F(a) \le u \le F(b)$, where $u = F(x)$. The calculations are done with an approximate precision of `tol`. Returns $x = F^{-1}(u)$. Restrictions: $u \in [0, 1]$.

```
public double inverseBisection (double u)
```

Computes and returns the inverse distribution function $x = F^{-1}(u)$, using bisection. Restrictions: $u \in [0, 1]$.

```
public double inverseF (double u)
```

Returns the inverse distribution function $x = F^{-1}(u)$. Restrictions: $u \in [0, 1]$.

```
public double getMean()
```

Returns the mean.

```
public double getVariance()
```

Returns the variance.

```
public double getStandardDeviation()
```

Returns the standard deviation.

```
public double getXinf()
```

Returns x_a such that the probability density is 0 everywhere outside the interval $[x_a, x_b]$.

`public double getXsup()`

Returns x_b such that the probability density is 0 everywhere outside the interval $[x_a, x_b]$.

`public void setXinf (double xa)`

Sets the value $x_a =$ `xa`, such that the probability density is 0 everywhere outside the interval $[x_a, x_b]$.

`public void setXsup (double xb)`

Sets the value $x_b =$ `xb`, such that the probability density is 0 everywhere outside the interval $[x_a, x_b]$.

DistributionFactory

This class implements a string API for the package `probdist`. It uses Java Reflection to allow the creation of probability distribution objects from a string. This permits one to obtain distribution specifications from a file or dynamically from user input during program execution. This string API is similar to that of UNURAN [36].

The (static) methods of this class invoke the constructor specified in the string. For example,

```
d = DistributionFactory.getContinuousDistribution ("NormalDist (0.0, 2.5)");
```

is equivalent to

```
d = NormalDist (0.0, 2.5);
```

The string that specifies the distribution (i.e., the formal parameter `str` of the methods) must be a valid call of the constructor of a class that extends `ContinuousDistribution` or `DiscreteDistribution`, and all parameter values must be numerical values (variable names are not allowed).

The distribution parameters can also be estimated from a set of observations instead of being passed to the constructor. In that case, one passes the vector of observations, and the constructor estimates the parameters by the maximum likelihood method.

```
package umontreal.iro.lecuyer.probdist;

public class DistributionFactory
```

```
public static ContinuousDistribution getDistributionMLE
                    (String distName, double[] x, int n)
```

Uses the Java Reflection API to construct a `ContinuousDistribution` object by estimating parameters of the distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static DiscreteDistributionInt getDistributionMLE
                    (String distName, int[] x, int n)
```

Uses the Java Reflection API to construct a `DiscreteDistributionInt` object by estimating parameters of the distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static <T extends ContinuousDistribution> T getDistributionMLE
                    (Class<T> distClass, double[] x, int n)
```

Uses the Java Reflection API to construct a `ContinuousDistribution` object by estimating parameters of the distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static <T extends DiscreteDistributionInt> T getDistributionMLE
                 (Class<T> distClass, int[] x, int n)
```

Uses the Java Reflection API to construct a `DiscreteDistributionInt` object by estimating parameters of the distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static ContinuousDistribution getContinuousDistribution (String str)
```

Uses the Java Reflection API to construct a `ContinuousDistribution` object by executing the code contained in the string `str`. This code should be a valid invocation of the constructor of a `ContinuousDistribution` object. This method throws exceptions if it cannot parse the given string and returns `null` if the distribution object could not be created due to a Java-specific instantiation problem.

```
public static DiscreteDistribution getDiscreteDistribution (String str)
```

Same as `getContinuousDistribution`, but for discrete distributions over the real numbers.

```
public static DiscreteDistributionInt getDiscreteDistributionInt (String str)
```

Same as `getContinuousDistribution`, but for discrete distributions over the integers.

InverseDistFromDensity

Implements a method for computing the inverse of an *arbitrary continuous* distribution function when only the probability density is known [14]. The cumulative probabilities (cdf) are pre-computed by numerical quadrature of the density using Gauss-Lobatto integration over suitably small intervals to satisfy the required precision, and these values are kept in tables. Then the algorithm uses polynomial interpolation over the tabulated values to get the inverse cdf. The user can select the desired precision and the degree of the interpolating polynomials.

The algorithm may fail for some distributions for which the density becomes infinite at a point (for ex. the Gamma and the Beta distributions with $\alpha < 1$) if one chooses too high a precision (a too small `eps`, for ex. $\epsilon \sim 10^{-15}$). However, it should work also for continuous densities with finite discontinuities.

While the setup time for this class is relatively slow, the numerical inversion is extremely fast and practically independent of the required precision and of the specific distribution. For comparisons between the times of standard inversion and inversion from this class as well as comparisons between setup times, see the introduction in class `InverseFromDensityGen` from package `randvar`.

Thus if only a few inverses are needed, then using this class is not efficient because of the slow set-up. But if one wants to call `inverseF` thousands of times or more, then using this class will be very efficient.

```
package umontreal.iro.lecuyer.probdist;
   import umontreal.iro.lecuyer.functions.MathFunction;


public class InverseDistFromDensity extends ContinuousDistribution
```

Constructors

```
public InverseDistFromDensity (ContinuousDistribution dist, double xc,
                               double eps, int order)
```

Given a continuous distribution `dist` with a well-defined density method, this class will compute tables for the numerical inverse of the distribution. The user may wish to set the left and the right boundaries between which the density is non-zero by calling methods `setXinf` and `setXsup` of `dist`, for better efficiency. Argument `xc` can be the mean, the mode or any other x for which the density is relatively large. The u-resolution `eps` is the required absolute error in the cdf, and `order` is the degree of the Newton interpolating polynomial over each interval. An `order` of 3 or 5, and an `eps` of 10^{-6} to 10^{-12} are usually good choices. Restrictions: $3 \le$ `order` ≤ 12.

```
public InverseDistFromDensity (MathFunction dens, double xc, double eps,
                               int order, double xleft, double xright)
```

Given a continuous probability density `dens`, this class will compute tables for the numerical inverse of the distribution. The left and the right boundaries of the density are `xleft` and `xright` (the density is 0 outside the interval `[xleft, xright]`). See the description of the other constructor.

Methods

```
public double density (double x)
```

Computes the probability density at x.

```
public double cdf (double x)
```

Computes the distribution function at x.

```
public double inverseF (double u)
```

Computes the inverse distribution function at u.

```
public double getXc()
```

Returns the `xc` given in the constructor.

```
public double getEpsilon()
```

Returns the u-resolution `eps` associated with this object.

```
public int getOrder()
```

Returns the order associated with this object.

```
public double[] getParams()
```

Return a table containing the parameters of the current distribution. This table is returned as: [`xc`, `eps`, `order`].

```
public String toString()
```

Returns a `String` containing information about the current distribution.

BernoulliDist

Extends the class `DiscreteDistributionInt` for the *Bernoulli* distribution [33] with parameter p, where $0 \le p \le 1$. Its mass function is given by

$$f(x) = \begin{cases} 1-p, & \text{if } x = 0; \\ p, & \text{if } x = 1; \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Its distribution function is

$$F(x) = \begin{cases} 0, & \text{if } x < 0; \\ 1-p, & \text{if } 0 \le x < 1; \\ 1, & \text{if } x \ge 1. \end{cases} \tag{7}$$

```
package umontreal.iro.lecuyer.probdist;

public class BernoulliDist extends DiscreteDistributionInt
```

Constructor

`public BernoulliDist (double p)`

Creates a Bernoulli distribution object.

Methods

`public static double prob (double p, int x)`

Returns the Bernoulli probability $f(x)$ with parameter p (see eq. (6)).

`public static double cdf (double p, int x)`

Returns the Bernoulli distribution function $F(x)$ with parameter p (see eq. (7)).

`public static double barF (double p, int x)`

Returns the complementary Bernoulli distribution function $\bar{F}(x) = P[X \ge x]$ with parameter p.

`public static int inverseF (double p, double u)`

Returns the inverse of the Bernoulli distribution function with parameter p at u.

`public static double[] getMLE (int[] x, int m)`

Estimates the parameters p of the Bernoulli distribution using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m-1$. The estimate is returned in a one-element array: $[p]$.

`public static BernoulliDist getInstanceFromMLE (int[] x, int m)`

Creates a new instance of a Bernoulli distribution with parameter p estimated using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m - 1$.

`public static double getMean (double p)`

Returns the mean $E[X] = p$ of the Bernoulli distribution with parameter p.

`public static double getVariance (double p)`

Computes the variance $\mathrm{Var}[X] = p(1 - p)$ of the Bernoulli distribution with parameter p.

`public static double getStandardDeviation (double p)`

Computes the standard deviation of the Bernoulli distribution with parameter p.

`public double getP()`

Returns the parameter p of this object.

`public double[] getParams ()`

Returns an array that contains the parameter p of the current distribution: $[p]$.

`public void setParams (double p)`

Resets the parameter to this new value.

BinomialDist

Extends the class `DiscreteDistributionInt` for the *binomial* distribution [33, page 321] with parameters n and p, where n is a positive integer and $0 \le p \le 1$. Its mass function is given by

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x} = \frac{n!}{x!(n-x)!}\, p^x (1-p)^{n-x} \qquad \text{for } x = 0, 1, 2, \ldots n, \tag{8}$$

and its distribution function is

$$F(x) = \sum_{j=0}^{x} \binom{n}{j}\, p^j (1-p)^{n-j} \qquad \text{for } x = 0, 1, 2, \ldots n, \tag{9}$$

```
package umontreal.iro.lecuyer.probdist;

public class BinomialDist extends DiscreteDistributionInt
```

Constructor

```
public BinomialDist (int n, double p)
```

Creates an object that contains the binomial terms (8), for $0 \le x \le n$, and the corresponding cumulative function. These values are computed and stored in dynamic arrays, unless n exceeds `MAXN`.

Methods

```
public static double prob (int n, double p, int x)
```

Computes and returns the binomial probability $p(x)$ in eq. (8).

```
public static double prob (int n, double p, double q, int x)
```

A generalization of the previous method. Computes and returns the binomial term

$$f(x) = \binom{n}{x} p^x q^{n-x} = \frac{n!}{x!(n-x)!}\, p^x q^{n-x}, \tag{10}$$

where p and q are arbitrary real numbers (q is not necessarily equal to $1-p$). In the case where $0 \le p \le 1$ and $q = 1-p$, the returned value is a probability term for the binomial distribution.

```
public static double cdf (int n, double p, int x)
```

Computes $F(x)$, the distribution function of a binomial random variable with parameters n and p, evaluated at x.

```
public static double barF (int n, double p, int x)
```

Returns $\bar{F}(x) = P[X \geq x]$, the complementary distribution function.

```
public static int inverseF (int n, double p, double u)
```

Computes $x = F^{-1}(u)$, the inverse of the binomial distribution.

```
public static double[] getMLE (int[] x, int m)
```

Estimates the parameters (n, p) of the binomial distribution using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m - 1$. The estimates are returned in a two-element array, in regular order: $[n, p]$.

```
public static BinomialDist getInstanceFromMLE (int[] x, int m)
```

Creates a new instance of a binomial distribution with both parameters n and p estimated using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m - 1$.

```
public static double[] getMLE (int[] x, int m, int n)
```

Estimates the parameter p of the binomial distribution with given (fixed) parameter n, by the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m - 1$. Returns the estimator in an array with a single element.

```
public static BinomialDist getInstanceFromMLE (int[] x, int m, int n)
```

Creates a new instance of a binomial distribution with given (fixed) parameter n, and with parameter p estimated by the maximum likelihood method based on the m observations $x[i]$, $i = 0, 1, \ldots, m - 1$.

```
public static double getMean (int n, double p)
```

Computes the mean $E[X] = np$ of the binomial distribution with parameters n and p.

```
public static double getVariance (int n, double p)
```

Computes the variance $\mathrm{Var}[X] = np(1 - p)$ of the binomial distribution with parameters n and p.

```
public static double getStandardDeviation (int n, double p)
```

Computes the standard deviation of the Binomial distribution with parameters n and p.

```
public int getN()
```

Returns the parameter n of this object.

```
public double getP()
```

Returns the parameter p of this object.

```
public double[] getParams ()
```

Returns a table that contains the parameters (n, p) of the current distribution, in regular order: $[n, p]$.

```
public void setParams (int n, double p)
```

Resets the parameters to these new values and recomputes everything as in the constructor. From the performance viewpoint, it is essentially the same as constructing a new `BinomialDist` object.

GeometricDist

Extends the class `DiscreteDistributionInt` for the *geometric* distribution [33, page 322] with parameter p, where $0 < p < 1$. Its mass function is

$$p(x) = p\,(1-p)^x, \qquad \text{for } x = 0, 1, 2, \ldots \tag{11}$$

The distribution function is given by

$$F(x) = 1 - (1-p)^{x+1}, \qquad \text{for } x = 0, 1, 2, \ldots \tag{12}$$

and its inverse is

$$F^{-1}(u) = \left\lfloor \frac{\ln(1-u)}{\ln(1-p)} \right\rfloor, \qquad \text{for } 0 \le u < 1. \tag{13}$$

```
package umontreal.iro.lecuyer.probdist;

public class GeometricDist extends DiscreteDistributionInt
```

Constructor

```
public GeometricDist (double p)
```

Constructs a geometric distribution with parameter p.

Methods

```
public static double prob (double p, int x)
```

Computes the geometric probability $p(x)$ given in (11) .

```
public static double cdf (double p, int x)
```

Computes the distribution function $F(x)$.

```
public static double barF (double p, int x)
```

Computes the complementary distribution function. *WARNING:* The complementary distribution function is defined as $\bar{F}(x) = P[X \ge x]$.

```
public static int inverseF (double p, double u)
```

Computes the inverse of the geometric distribution, given by (13).

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

Estimates the parameter p of the geometric distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimate is returned in element 0 of the returned array.

`public static GeometricDist getInstanceFromMLE (int[] x, int n)`

Creates a new instance of a geometric distribution with parameter p estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double p)`

Computes and returns the mean $E[X] = (1-p)/p$ of the geometric distribution with parameter p.

`public static double getVariance (double p)`

Computes and returns the variance $\mathrm{Var}[X] = (1-p)/p^2$ of the geometric distribution with parameter p.

`public static double getStandardDeviation (double p)`

Computes and returns the standard deviation of the geometric distribution with parameter p.

`public double getP()`

Returns the p associated with this object.

`public void setP (double p)`

Resets the value of p associated with this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution.

HypergeometricDist

Extends the class `DiscreteDistributionInt` for the *hypergeometric* distribution [20, page 101] with k elements chosen among l, m being of one type, and $l - m$ of the other. The parameters m, k and l are positive integers where $1 \le m \le l$ and $1 \le k \le l$. Its mass function is given by

$$p(x) = \frac{\binom{m}{x}\binom{l-m}{k-x}}{\binom{l}{k}} \qquad \text{for } \max(0, k-l+m) \le x \le \min(k, m). \tag{14}$$

```
package umontreal.iro.lecuyer.probdist;

public class HypergeometricDist extends DiscreteDistributionInt
```

Constructor

`public HypergeometricDist (int m, int l, int k)`

Constructs an hypergeometric distribution with parameters m, l and k.

Methods

`public static double prob (int m, int l, int k, int x)`

Computes the hypergeometric probability $p(x)$ given by (14).

`public static double cdf (int m, int l, int k, int x)`

Computes the distribution function $F(x)$.

`public static double barF (int m, int l, int k, int x)`

Computes the complementary distribution function. *WARNING:* The complementary distribution function is defined as $\bar{F}(x) = P[X \ge x]$.

`public static int inverseF (int m, int l, int k, double u)`

Computes $F^{-1}(u)$ for the hypergeometric distribution without using precomputed tables. The inversion is computed using the chop-down algorithm [28].

`public static double getMean (int m, int l, int k)`

Computes and returns the mean $E[X] = km/l$ of the Hypergeometric distribution with parameters m, l and k.

`public static double getVariance (int m, int l, int k)`

Computes and returns the variance $\mathrm{Var}[X] = \frac{(km/l)(1-m/l)(l-k)}{l-1}$ of the hypergeometric distribution with parameters m, l and k.

`public static double getStandardDeviation (int m, int l, int k)`

Computes and returns the standard deviation of the hypergeometric distribution with parameters m, l and k.

`public int getM()`

Returns the m associated with this object.

`public int getL()`

Returns the l associated with this object.

`public int getK()`

Returns the k associated with this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[m, l, k]$.

`public void setParams (int m, int l, int k)`

Resets the parameters of this object to m, l and k.

LogarithmicDist

Extends the class `DiscreteDistributionInt` for the *logarithmic* distribution. It has shape parameter θ, where $0 < \theta < 1$. Its mass function is

$$p(x) = \frac{-\theta^x}{x \log(1-\theta)} \qquad \text{for } x = 1, 2, 3, \ldots \tag{15}$$

Its distribution function is

$$F(x) = \frac{-1}{\log(1-\theta)} \sum_{i=1}^{x} \frac{\theta^i}{i}, \qquad \text{for } x = 1, 2, 3, \ldots \tag{16}$$

and is 0 for $x \le 0$.

```
package umontreal.iro.lecuyer.probdist;

public class LogarithmicDist extends DiscreteDistributionInt
```

Constructor

`public LogarithmicDist (double theta)`

Constructs a logarithmic distribution with parameter $\theta =$ `theta`.

Methods

`public static double prob (double theta, int x)`

Computes the logarithmic probability $p(x)$ given in (15) .

`public static double cdf (double theta, int x)`

Computes the distribution function $F(x)$.

`public static double barF (double theta, int x)`

Computes the complementary distribution function. *WARNING:* The complementary distribution function is defined as $\bar{F}(x) = P[X \ge x]$.

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

Estimates the parameter θ of the logarithmic distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimate is returned in element 0 of the returned array.

`public static LogarithmicDist getInstanceFromMLE (int[] x, int n)`

Creates a new instance of a logarithmic distribution with parameter θ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double theta)`

Computes and returns the mean

$$E[X] = \frac{-\theta}{(1-\theta)\ln(1-\theta)}$$

of the logarithmic distribution with parameter $\theta =$ `theta`.

`public static double getVariance (double theta)`

Computes and returns the variance

$$\mathrm{Var}[X] = \frac{-\theta(\theta + \ln(1-\theta))}{[(1-\theta)\ln(1-\theta)]^2}$$

of the logarithmic distribution with parameter $\theta =$ `theta`.

`public static double getStandardDeviation (double theta)`

Computes and returns the standard deviation of the logarithmic distribution with parameter $\theta =$ `theta`.

`public double getTheta()`

Returns the θ associated with this object.

`public void setTheta (double theta)`

Sets the θ associated with this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution.

NegativeBinomialDist

Extends the class `DiscreteDistributionInt` for the *negative binomial* distribution [33, page 324] with real parameters n and p, where $n > 0$ and $0 \le p \le 1$. Its mass function is

$$p(x) = \frac{\Gamma(n+x)}{\Gamma(n)\, x!} p^n (1-p)^x, \qquad \text{for } x = 0, 1, 2, \ldots \tag{17}$$

where $\Gamma(x)$ is the gamma function.

If n is an integer, $p(x)$ can be interpreted as the probability of having x failures before the n-th success in a sequence of independent Bernoulli trials with probability of success p. This special case is implemented as the Pascal distribution (see `PascalDist`).

```
package umontreal.iro.lecuyer.probdist;

public class NegativeBinomialDist extends DiscreteDistributionInt
```

Constructor

`public NegativeBinomialDist (double n, double p)`

Creates an object that contains the probability terms (17) and the distribution function for the negative binomial distribution with parameters n and p.

Methods

`public static double prob (double n, double p, int x)`

Computes the probability $p(x)$ defined in (17).

`public static double cdf (double n, double p, int x)`

Computes the distribution function.

`public static double barF (double n, double p, int x)`

Returns $\bar{F}(x) = P[X \ge x]$, the complementary distribution function.

`public static int inverseF (double n, double p, double u)`

Computes the inverse function without precomputing tables.

`public static double[] getMLE (int[] x, int m, double n)`

Estimates the parameter p of the negative binomial distribution using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m-1$. The parameter n is assumed known. The estimate $\hat{p}$ is returned in element 0 of the returned array. The maximum likelihood estimator $\hat{p}$ satisfies the equation $\hat{p} = n/(n + \bar{x}_m)$, where $\bar{x}_m$ is the average of $x[0], \ldots, x[m-1]$.

```
public static NegativeBinomialDist getInstanceFromMLE (int[] x, int m,
                                                        double n)
```

Creates a new instance of a negative binomial distribution with parameters n given and $\hat{p}$ estimated using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m-1$.

```
public static double[] getMLE1 (int[] x, int m, double p)
```

Estimates the parameter n of the negative binomial distribution using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m-1$. The parameter p is assumed known. The estimate $\hat{n}$ is returned in element 0 of the returned array.

```
public static NegativeBinomialDist getInstanceFromMLE1 (int[] x, int m,
                                                         double p)
```

Creates a new instance of a negative binomial distribution with parameters p given and $\hat{n}$ estimated using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m-1$.

```
public static double[] getMLE (int[] x, int m)
```

Estimates the parameter (n, p) of the negative binomial distribution using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m-1$. The estimates are returned in a two-element array, in regular order: [n, p].

```
public static NegativeBinomialDist getInstanceFromMLE (int[] x, int m)
```

Creates a new instance of a negative binomial distribution with parameters n and p estimated using the maximum likelihood method based on the m observations $x[i]$, $i = 0, 1, \ldots, m-1$.

```
public static double getMean (double n, double p)
```

Computes and returns the mean $E[X] = n(1-p)/p$ of the negative binomial distribution with parameters n and p.

```
public static double getVariance (double n, double p)
```

Computes and returns the variance $\mathrm{Var}[X] = n(1-p)/p^2$ of the negative binomial distribution with parameters n and p.

```
public static double getStandardDeviation (double n, double p)
```

Computes and returns the standard deviation of the negative binomial distribution with parameters n and p.

```
public double getN()
```

Returns the parameter n of this object.

```
public double getP()
```

Returns the parameter p of this object.

```
public void setParams (double n, double p)
```

Sets the parameter n and p of this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[n, p]$.

PascalDist

The *Pascal* distribution is a special case of the *negative binomial* distribution [33, page 324] with parameters n and p, where n is a positive integer and $0 \le p \le 1$. Its mass function is

$$p(x) = \binom{n+x-1}{x} p^n (1-p)^x, \qquad \text{for } x = 0, 1, 2, \ldots \tag{18}$$

This $p(x)$ can be interpreted as the probability of having x failures before the nth success in a sequence of independent Bernoulli trials with probability of success p. For $n = 1$, this gives the *geometric* distribution.

```
package umontreal.iro.lecuyer.probdist;

public class PascalDist extends NegativeBinomialDist
```

Constructor

```
public PascalDist (int n, double p)
```

Creates an object that contains the probability terms (18) and the distribution function for the Pascal distribution with parameter n and p.

Methods

```
public static double[] getMLE (int[] x, int m)
```

Estimates the parameter (n, p) of the Pascal distribution using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m-1$. The estimates are returned in a two-element array, in regular order: $[n, p]$.

```
public static PascalDist getInstanceFromMLE (int[] x, int m)
```

Creates a new instance of a Pascal distribution with parameters n and p estimated using the maximum likelihood method based on the m observations $x[i]$, $i = 0, 1, \ldots, m-1$.

```
public int getN1()
```

Returns the parameter n of this object.

```
public void setParams (int n, double p)
```

Sets the parameter n and p of this object.

PoissonDist

Extends the class `DiscreteDistributionInt` for the *Poisson* distribution [33, page 325] with mean $\lambda \geq 0$. The mass function is

$$p(x) \;=\; \frac{e^{-\lambda}\lambda^x}{x!}, \qquad \text{for } x = 0, 1, \ldots \tag{19}$$

and the distribution function is

$$F(x) \;=\; e^{-\lambda}\sum_{j=0}^{x}\frac{\lambda^j}{j!}, \qquad \text{for } x = 0, 1, \ldots. \tag{20}$$

If one has to compute $p(x)$ and/or $F(x)$ for several values of x with the same λ, where λ is not too large, then it is more efficient to instantiate an object and use the non-static methods, since the functions will then be computed once and kept in arrays.

For the static methods that compute $F(x)$ and $\bar{F}(x)$, we exploit the relationship $F(x) = 1 - G_{x+1}(\lambda)$, where G_{x+1} is the *gamma* distribution function with parameters $(\alpha, \lambda) = (x + 1, 1)$.

```
package umontreal.iro.lecuyer.probdist;

public class PoissonDist extends DiscreteDistributionInt
```

Constructor

`public PoissonDist (double lambda)`

Creates an object that contains the probability and distribution functions, for the Poisson distribution with parameter `lambda`, which are computed and stored in dynamic arrays inside that object.

Methods

`public static double prob (double lambda, int x)`

Computes and returns the Poisson probability $p(x)$ for $\lambda =$ `lambda`, as defined in (19).

`public static double cdf (double lambda, int x)`

Computes and returns the value of the Poisson distribution function $F(x)$ for $\lambda =$ `lambda`, as defined in (20).

`public static double barF (double lambda, int x)`

Computes and returns the value of the complementary Poisson distribution function, for $\lambda =$ `lambda`. *WARNING:* The complementary distribution function is defined as $\bar{F}(x) = P[X \geq x]$.

`public static int inverseF (double lambda, double u)`

Performs a linear search to get the inverse function without precomputed tables.

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

Estimates the parameter λ of the Poisson distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The maximum likelihood estimator $\hat{\lambda}$ satisfy the equation $\hat{\lambda} = \bar{x}_n$, where $\bar{x}_n$ is the average of $x[0], \ldots, x[n-1]$ (see [33, page 326]).

`public static PoissonDist getInstanceFromMLE (int[] x, int n)`

Creates a new instance of a Poisson distribution with parameter λ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double lambda)`

Computes and returns the mean $E[X] = \lambda$ of the Poisson distribution with parameter λ.

`public static double getVariance (double lambda)`

Computes and returns the variance $= \lambda$ of the Poisson distribution with parameter λ.

`public static double getStandardDeviation (double lambda)`

Computes and returns the standard deviation of the Poisson distribution with parameter λ.

`public double getLambda()`

Returns the λ associated with this object.

`public void setLambda (double lambda)`

Sets the λ associated with this object.

`public double[] getParams ()`

Return a table containing the parameter of the current distribution.

UniformIntDist

Extends the class `DiscreteDistributionInt` for the *discrete uniform* distribution over the range $[i, j]$. Its mass function is given by

$$p(x) = \frac{1}{j - i + 1} \qquad \text{for } x = i, i + 1, \ldots, j \tag{21}$$

and 0 elsewhere. The distribution function is

$$F(x) = \begin{cases} 0, & \text{for } x < i \\ \dfrac{\lfloor x \rfloor - i + 1}{j - i + 1}, & \text{for } i \le x < j \\ 1, & \text{for } x \ge j. \end{cases} \tag{22}$$

and its inverse is

$$F^{-1}(u) = i + \lfloor (j - i + 1) u \rfloor \qquad \text{for } 0 \le u \le 1. \tag{23}$$

```
package umontreal.iro.lecuyer.probdist;

public class UniformIntDist extends DiscreteDistributionInt
```

Constructor

`public UniformIntDist (int i, int j)`

Constructs a discrete uniform distribution over the interval $[i, j]$.

Methods

`public static double prob (int i, int j, int x)`

Computes the discrete uniform probability $p(x)$ defined in (21).

`public static double cdf (int i, int j, int x)`

Computes the discrete uniform distribution function defined in (22).

`public static double barF (int i, int j, int x)`

Computes the discrete uniform complementary distribution function $\bar{F}(x)$. *WARNING:* The complementary distribution function is defined as $\bar{F}(x) = P[X \ge x]$.

`public static int inverseF (int i, int j, double u)`

Computes the inverse of the discrete uniform distribution function (23).

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

Estimates the parameters (i, j) of the uniform distribution over integers using the maximum likelihood method, from the n observations $x[k]$, $k = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[i,\ j]$.

`public static UniformIntDist getInstanceFromMLE (int[] x, int n)`

Creates a new instance of a discrete uniform distribution over integers with parameters i and j estimated using the maximum likelihood method based on the n observations $x[k]$, $k = 0, 1, \ldots, n-1$.

`public static double getMean (int i, int j)`

Computes and returns the mean $E[X] = (i + j)/2$ of the discrete uniform distribution.

`public static double getVariance (int i, int j)`

Computes and returns the variance $\mathrm{Var}[X] = [(j - i + 1)^2 - 1]/12$ of the discrete uniform distribution.

`public static double getStandardDeviation (int i, int j)`

Computes and returns the standard deviation of the discrete uniform distribution.

`public int getI()`

Returns the parameter i.

`public int getJ()`

Returns the parameter j.

`public void setParams (int i, int j)`

Sets the parameters i and j for this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[i,\ j]$.

ConstantIntDist

Extends the class `UniformIntDist` for a *constant* discrete distribution taking a single integer value with probability 1. Its mass function is

$$p(x) = \begin{cases} 1, & \text{for } x = c, \\ 0, & \text{elsewhere.} \end{cases} \tag{24}$$

Its distribution function is

$$F(x) = \begin{cases} 0, & \text{for } x < c \\ 1, & \text{for } x \ge c. \end{cases} \tag{25}$$

```
package umontreal.iro.lecuyer.probdist;

public class ConstantIntDist extends UniformIntDist
```

Constructor

```
public ConstantIntDist (int c)
```

Constructs a new constant distribution with probability 1 at `c`.

DiscreteDistribution

This class implements discrete distributions over a *finite set of real numbers* (also over *integers* as a particular case). We assume that the random variable X of interest can take one of the n values $x_0 < \cdots < x_{n-1}$, which *must be sorted* by increasing order. X can take the value x_k with probability $p_k = P[X = x_k]$. In addition to the methods specified in the interface `Distribution`, a method that returns the probability p_k is supplied.

```
package umontreal.iro.lecuyer.probdist;

public class DiscreteDistribution implements Distribution
```

Constructors

```
public DiscreteDistribution (double[] values, double[] prob, int n)
```

Constructs a discrete distribution over the n values contained in array `values`, with probabilities given in array `prob`. Both arrays must have at least n elements, the probabilities must sum to 1, and the values are assumed to be sorted by increasing order.

```
public DiscreteDistribution (int[] values, double[] prob, int n)
```

Similar to `DiscreteDistribution(double[], double[], int)`.

```
@Deprecated
public DiscreteDistribution (double[] params)
```

Constructs a discrete distribution whose parameters are given in a single ordered array: `params[0]` contains n, the number of values to consider. Then the next n values of `params` are the values, and the last n values of `params` are the probabilities values.

Methods

```
public double getMean()
```

Computes the mean $E[X] = \sum_i p_i x_i$ of the distribution.

```
public double getVariance()
```

Computes the variance $\mathrm{Var}[X] = \sum_i p_i (x_i - E[X])^2$ of the distribution.

```
public double getStandardDeviation()
```

Computes the standard deviation of the distribution.

```
public double[] getParams()
```

Returns a table containing the parameters of the current distribution. This table is built in regular order, according to constructor `DiscreteDistribution(double[] params)` order.

`public int getN()`

Returns the number of possible values x_i.

`public double prob (int i)`

Returns p_i, the probability of the i-th value, for $0 \le i < n$.

`public double getValue (int i)`

Returns the i-th value x_i, for $0 \le i < n$.

`public double getXinf()`

Returns the lower limit x_0 of the support of the distribution.

`public double getXsup()`

Returns the upper limit x_{n-1} of the support of the distribution.

`public String toString()`

Returns a `String` containing information about the current distribution.

ConstantDist

Extends the class `DiscreteDistribution` for a *constant* discrete distribution taking a single real value with probability 1. Its mass function is

$$p(x) = \begin{cases} 1, & \text{for } x = c, \\ 0, & \text{elsewhere.} \end{cases} \tag{26}$$

Its distribution function is

$$F(x) = \begin{cases} 0, & \text{for } x < c \\ 1, & \text{for } x \geq c. \end{cases} \tag{27}$$

```
package umontreal.iro.lecuyer.probdist;

public class ConstantDist extends DiscreteDistribution
```

Constructor

```
public ConstantDist (double c)
```

Constructs a new constant distribution with probability 1 at `c`.

EmpiricalDist

Extends `DiscreteDistribution` to an *empirical* distribution function, based on the observations $X_{(1)}, \dots, X_{(n)}$ (sorted by increasing order). The distribution is uniform over the n observations, so the distribution function has a jump of $1/n$ at each of the n observations.

```
package umontreal.iro.lecuyer.probdist;

public class EmpiricalDist extends DiscreteDistribution
```

Constructors

`public EmpiricalDist (double[] obs)`

Constructs a new empirical distribution using all the observations stored in `obs`, and which are assumed to have been sorted in increasing numerical order. [1] These observations are copied into an internal array.

`public EmpiricalDist (Reader in) throws IOException`

Constructs a new empirical distribution using the observations read from the reader `in`. This constructor will read the first `double` of each line in the stream. Any line that does not start with a `+`, `-`, or a decimal digit, is ignored. One must be careful about lines starting with a blank. This format is the same as in UNURAN. The observations read are assumed to have been sorted in increasing numerical order.

Methods

`public double getMedian ()`

Returns the median. Returns the $n/2^{\text{th}}$ item of the sorted observations when the number of items is odd, and the mean of the $n/2^{\text{th}}$ and the $(n/2+1)^{\text{th}}$ items when the number of items is even.

`public static double getMedian (double obs[], int n)`

Returns the median. Returns the $n/2^{\text{th}}$ item of the array `obs` when the number of items is odd, and the mean of the $n/2^{\text{th}}$ and the $(n/2+1)^{\text{th}}$ items when the number of items is even. The array does not have to be sorted.

`public int getN()`

Returns n, the number of observations.

`public double getObs (int i)`

Returns the value of $X_{(i)}$, for $i = 0, 1, \dots, n-1$.

[1] The method `java.util.Arrays.sort` may be used to sort the observations.

`public double getSampleMean()`

Returns the sample mean of the observations.

`public double getSampleVariance()`

Returns the sample variance of the observations.

`public double getSampleStandardDeviation()`

Returns the sample standard deviation of the observations.

`public double getInterQuartileRange()`

Returns the *interquartile range* of the observations, defined as the difference between the third and first quartiles.

`public double[] getParams ()`

Return a table containing parameters of the current distribution.

`public String toString ()`

Returns a `String` containing information about the current distribution.

BetaDist

Extends the class `ContinuousDistribution` for the *beta* distribution [27, page 210] with shape parameters $\alpha > 0$ and $\beta > 0$, over the interval (a, b), where $a < b$. This distribution has density

$$f(x) = \frac{(x-a)^{\alpha-1}(b-x)^{\beta-1}}{\mathcal{B}(\alpha,\beta)(b-a)^{\alpha+\beta-1}}, \qquad \text{for } a < x < b, \text{ and 0 elsewhere,} \tag{28}$$

and distribution function

$$F(x) = I_{\alpha,\beta}(x) = \int_a^x \frac{(\xi-a)^{\alpha-1}(b-\xi)^{\beta-1}}{\mathcal{B}(\alpha,\beta)(b-a)^{\alpha+\beta-1}} d\xi, \qquad \text{for } a < x < b, \tag{29}$$

where $\mathcal{B}(\alpha, \beta)$ is the *beta* function defined by

$$\mathcal{B}(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}, \tag{30}$$

and $\Gamma(x)$ is the gamma function defined in (50).

```
package  umontreal.iro.lecuyer.probdist;

public class BetaDist extends ContinuousDistribution
```

Constructors

`public BetaDist (double alpha, double beta)`

Constructs a `BetaDist` object with parameters $\alpha =$ `alpha` and $\beta =$ `beta` and default domain $(0, 1)$.

`public BetaDist (double alpha, double beta, double a, double b)`

Constructs a `BetaDist` object with parameters $\alpha =$ `alpha` and $\beta =$ `beta`, and domain (`a`, `b`).

`public BetaDist (double alpha, double beta, int d)`

Constructs a `BetaDist` object with parameters $\alpha =$ `alpha` and $\beta =$ `beta`, and approximations of roughly `d` decimal digits of precision when computing the distribution, complementary distribution, and inverse functions. The default domain $(0, 1)$ is used.

`public BetaDist (double alpha, double beta, double a, double b, int d)`

Constructs a `BetaDist` object with parameters $\alpha =$ `alpha` and $\beta =$ `beta`, and approximations of roughly `d` decimal digits of precision when computing distribution, complementary distribution, and inverse functions. The domain (`a`, `b`) is used.

Methods

```
public static double density (double alpha, double beta, double x)
```

Same as `density (alpha, beta, 0, 1, x)`.

```
public static double density (double alpha, double beta,
                              double a, double b, double x)
```

Computes the density function of the *beta* distribution.

```
public static double cdf (double alpha, double beta, int d, double x)
```

Same as `cdf (alpha, beta, 0, 1, d, x)`.

```
public static double cdf (double alpha, double beta,
                          double a, double b, int d, double x)
```

Computes an approximation of the distribution function, with roughly `d` decimal digits of precision.

```
public static double barF (double alpha, double beta, int d, double x)
```

Same as `barF (alpha, beta, 0, 1, d, x)`.

```
public static double barF (double alpha, double beta,
                           double a, double b, int d, double x)
```

Computes the complementary distribution function.

```
public static double inverseF (double alpha, double beta, int d, double u)
```

Same as `inverseF (alpha, beta, 0, 1, d, u)`.

```
public static double inverseF (double alpha, double beta,
                               double a, double b, int d, double u)
```

Returns the inverse beta distribution function using the algorithm implemented in [42]. The method performs interval halving or Newton iterations to compute the inverse. The precision depends on the accuracy of the `cdf` method. The argument `d` gives a good idea of the precision attained.

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

Estimates the parameters (α, β) of the beta distribution over the interval $[0, 1]$ using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\alpha, \beta]$.

```
public static BetaDist getInstanceFromMLE (double[] x, int n)
```

Creates a new instance of a beta distribution with parameters α and β over the interval $[0, 1]$ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static double getMean (double alpha, double beta)
```

Computes and returns the mean $E[X] = \alpha/(\alpha+\beta)$ of the beta distribution with parameters α and β, over the interval $[0, 1]$.

```
public static double getMean (double alpha, double beta, double a,
                              double b)
```

Computes and returns the mean $E[X] = (b\alpha + a\beta)/(\alpha + \beta)$ of the beta distribution with parameters α and β over the interval $[a, b]$.

```
public static double getVariance (double alpha, double beta)
```

Computes and returns the variance $\mathrm{Var}[X] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$ of the beta distribution with parameters α and β, over the interval $[0, 1]$.

```
public static double getVariance (double alpha, double beta, double a,
                                  double b)
```

Computes and returns the variance $\mathrm{Var}[X] = \frac{\alpha\beta(b-a)^2}{(\alpha+\beta)^2(\alpha+\beta+1)}$ of the beta distribution with parameters α and β, over the interval $[a, b]$.

```
public static double getStandardDeviation (double alpha, double beta)
```

Computes the standard deviation of the beta distribution with parameters α and β, over the interval $[0, 1]$.

```
public static double getStandardDeviation (double alpha, double beta,
                                           double a, double b)
```

Computes the standard deviation of the beta distribution with parameters α and β, over the interval $[a, b]$.

```
public double getAlpha()
```

Returns the parameter α of this object.

```
public double getBeta()
```

Returns the parameter β of this object.

```
public double getA()
```

Returns the parameter a of this object.

```
public double getB()
```

Returns the parameter b of this object.

```
public double[] getParams ()
```

Return a table containing parameters of the current distribution. This table is put in regular order: $[\alpha,\ \beta]$.

BetaSymmetricalDist

Specializes the class `BetaDist` to the case of a *symmetrical beta* distribution over the interval $[0, 1]$, with shape parameters $\alpha = \beta$. A faster inversion method is implemented here for this special case. Because of the symmetry around 1/2, four series are used to compute the `cdf`, two around $x = 0$ and two around $x = 1/2$. Given u, one then solves each series for x by using the Newton-Raphson method which shows quadratic convergence when the starting iterate is close enough to the solution x.

```
package  umontreal.iro.lecuyer.probdist;

public class BetaSymmetricalDist extends BetaDist
```

Constructors

```
public BetaSymmetricalDist (double alpha)
```

Constructs a `BetaSymmetricalDist` object with parameters $\alpha = \beta =$ `alpha`, over the unit interval $(0, 1)$.

```
public BetaSymmetricalDist (double alpha, int d)
```

Same as `BetaSymmetricalDist (alpha)`, but using approximations of roughly `d` decimal digits of precision when computing the distribution, complementary distribution, and inverse functions.

Methods

```
public static double density (double alpha, double x)
```

Returns the density evaluated at x.

```
public static double cdf (double alpha, int d, double x)
```

Same as `cdf (alpha, alpha, d, x)`.

```
public static double barF (double alpha, int d, double x)
```

Returns the complementary distribution function.

```
public static double inverseF (double alpha, double u)
```

Returns the inverse distribution function evaluated at u, for the symmetrical beta distribution over the interval $[0, 1]$, with shape parameters $0 < \alpha = \beta =$ `alpha`. Uses four different hypergeometric series to compute the distribution $u = F(x)$ (for the four cases x close to 0 and $\alpha < 1$, x close to 0 and $\alpha > 1$, x close to 1/2 and $\alpha < 1$, and x close to 1/2 and $\alpha > 1$), which are then solved by Newton's method for the solution of equations. For $\alpha > 100000$, uses a normal approximation given in [46].

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

Estimates the parameter α of the symmetrical beta distribution over the interval [0, 1] using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimate is returned in element 0 of the returned array.

```
public static BetaSymmetricalDist getInstanceFromMLE (double[] x, int n)
```

Creates a new instance of a symmetrical beta distribution with parameter α estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static double getMean (double alpha)
```

Computes and returns the mean $E[X] = 1/2$ of the symmetrical beta distribution with parameter α.

```
public static double getVariance (double alpha)
```

Computes and returns the variance, $\mathrm{Var}[X] = 1/(8\alpha + 4)$, of the symmetrical beta distribution with parameter α.

```
public static double getStandardDeviation (double alpha)
```

Computes and returns the standard deviation of the symmetrical beta distribution with parameter α.

```
public double[] getParams ()
```

Return a table containing the parameter of the current distribution.

CauchyDist

Extends the class `ContinuousDistribution` for the *Cauchy* distribution [26, page 299] with location parameter α and scale parameter $\beta > 0$. The density function is given by

$$f(x) = \frac{\beta}{\pi[(x-\alpha)^2 + \beta^2]}, \qquad \text{for } -\infty < x < \infty. \tag{31}$$

The distribution function is

$$F(x) = \frac{1}{2} + \frac{\arctan((x-\alpha)/\beta)}{\pi}, \qquad \text{for } -\infty < x < \infty, \tag{32}$$

and its inverse is

$$F^{-1}(u) = \alpha + \beta \tan(\pi(u - 1/2)). \qquad \text{for } 0 < u < 1. \tag{33}$$

```
package umontreal.iro.lecuyer.probdist;

public class CauchyDist extends ContinuousDistribution
```

Constructors

`public CauchyDist()`

Constructs a `CauchyDist` object with parameters $\alpha = 0$ and $\beta = 1$.

`public CauchyDist (double alpha, double beta)`

Constructs a `CauchyDist` object with parameters $\alpha =$ `alpha` and $\beta =$ `beta`.

Methods

`public static double density (double alpha, double beta, double x)`

Computes the density function.

`public static double cdf (double alpha, double beta, double x)`

Computes the distribution function.

`public static double barF (double alpha, double beta, double x)`

Computes the complementary distribution.

`public static double inverseF (double alpha, double beta, double u)`

Computes the inverse of the distribution.

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

Estimates the parameters (α, β) of the Cauchy distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\alpha, \beta]$.

`public static CauchyDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a Cauchy distribution with parameters α and β estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double alpha, double beta)`

Throws an exception since the mean does not exist.

`public static double getVariance (double alpha, double beta)`

Returns ∞ since the variance does not exist.

`public static double getStandardDeviation (double alpha, double beta)`

Returns ∞ since the standard deviation does not exist.

`public double getAlpha()`

Returns the value of α for this object.

`public double getBeta()`

Returns the value of β for this object.

`public void setParams (double alpha, double beta)`

Sets the value of the parameters α and β for this object.

`public double[] getParams ()`

Return a table containing parameters of the current distribution. This table is put in regular order: $[\alpha, \beta]$.

ChiDist

Extends the class `ContinuousDistribution` for the *chi* distribution [26, page 417] with shape parameter $\nu > 0$, where the number of degrees of freedom ν is a positive integer. The density function is given by

$$f(x) = \frac{e^{-x^2/2} x^{\nu-1}}{2^{(\nu/2)-1}\Gamma(\nu/2)}, \qquad \text{for } x > 0, \tag{34}$$

where $\Gamma(x)$ is the gamma function defined in (50). The distribution function is

$$F(x) = \frac{1}{\Gamma(\nu/2)} \int_0^{x^2/2} t^{\nu/2-1} e^{-t}\, dt. \tag{35}$$

It is equivalent to the gamma distribution function with parameters $\alpha = \nu/2$ and $\lambda = 1$, evaluated at $x^2/2$.

```
package umontreal.iro.lecuyer.probdist;

public class ChiDist extends ContinuousDistribution
```

Constructor

```
public ChiDist (int nu)
```

Constructs a `ChiDist` object.

Methods

```
public static double density (int nu, double x)
```

Computes the density function.

```
public static double cdf (int nu, double x)
```

Computes the distribution function by using the gamma distribution function.

```
public static double barF (int nu, double x)
```

Computes the complementary distribution.

```
public static double inverseF (int nu, double u)
```

Returns the inverse distribution function computed using the gamma inversion.

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

Estimates the parameter ν of the chi distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimate is returned in element 0 of the returned array.

`public static ChiDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a chi distribution with parameter ν estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (int nu)`

Computes and returns the mean

$$E[X] = \frac{\sqrt{2}\,\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})}$$

of the chi distribution with parameter ν.

`public static double getVariance (int nu)`

Computes and returns the variance

$$\mathrm{Var}[X] = \frac{2\,\Gamma(\frac{\nu}{2})\Gamma(1+\frac{\nu}{2}) - \Gamma^2(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})}$$

of the chi distribution with parameter ν.

`public static double getStandardDeviation (int nu)`

Computes and returns the standard deviation of the chi distribution with parameter ν.

`public int getNu()`

Returns the value of ν for this object.

`public void setNu (int nu)`

Sets the value of ν for this object.

`public double[] getParams ()`

Return a table containing parameters of the current distribution.

ChiSquareDist

Extends the class `ContinuousDistribution` for the *chi-square* distribution with n degrees of freedom, where n is a positive integer [26, page 416]. Its density is

$$f(x) = \frac{x^{(n/2)-1}e^{-x/2}}{2^{n/2}\Gamma(n/2)}, \qquad \text{for } x > 0 \tag{36}$$

where $\Gamma(x)$ is the gamma function defined in (50). The *chi-square* distribution is a special case of the *gamma* distribution with shape parameter $n/2$ and scale parameter $1/2$. Therefore, one can use the methods of `GammaDist` for this distribution.

The non-static versions of the methods `cdf`, `barF`, and `inverseF` call the static version of the same name.

```
package umontreal.iro.lecuyer.probdist;

public class ChiSquareDist extends ContinuousDistribution
```

Constructor

`public ChiSquareDist (int n)`

Constructs a chi-square distribution with `n` degrees of freedom.

Methods

`public static double density (int n, double x)`

Computes the density function (36) for a *chi-square* distribution with n degrees of freedom.

`public static double cdf (int n, int d, double x)`

Computes the chi-square distribution function with n degrees of freedom, evaluated at x. The method tries to return d decimals digits of precision, but there is no guarantee.

`public static double barF (int n, int d, double x)`

Computes the complementary chi-square distribution function with n degrees of freedom, evaluated at x. The method tries to return d decimals digits of precision, but there is no guarantee.

`public static double inverseF (int n, double u)`

Computes an approximation of $F^{-1}(u)$, where F is the chi-square distribution with n degrees of freedom. Uses the approximation given in [2] and in Figure L.23 of [7]. It gives at least 6 decimal digits of precision, except far in the tails (that is, for $u < 10^{-5}$ or $u > 1 - 10^{-5}$) where the function calls the method `GammaDist.inverseF (n/2, 7, u)` and multiplies the result by 2.0. To get better precision, one may call `GammaDist.inverseF`, but this method is slower than the current method, especially for large n. For instance, for $n = 16$, 1024, and 65536, the `GammaDist.inverseF` method is 2, 5, and 8 times slower, respectively, than the current method.

`public static double[] getMLE (double[] x, int m)`

Estimates the parameter n of the chi-square distribution using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m - 1$. The estimate is returned in element 0 of the returned array.

`public static ChiSquareDist getInstanceFromMLE (double[] x, int m)`

Creates a new instance of a chi-square distribution with parameter n estimated using the maximum likelihood method based on the m observations $x[i]$, $i = 0, 1, \ldots, m - 1$.

`public static double getMean (int n)`

Computes and returns the mean $E[X] = n$ of the chi-square distribution with parameter n.

`public static double[] getMomentsEstimate (double[] x, int m)`

Estimates and returns the parameter $[\hat{n}]$ of the chi-square distribution using the moments method based on the m observations in table $x[i]$, $i = 0, 1, \ldots, m - 1$.

`public static double getVariance (int n)`

Returns the variance $\mathrm{Var}[X] = 2n$ of the chi-square distribution with parameter n.

`public static double getStandardDeviation (int n)`

Returns the standard deviation of the chi-square distribution with parameter n.

`public int getN()`

Returns the parameter n of this object.

`public void setN (int n)`

Sets the parameter n of this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution.

ChiSquareDistQuick

Provides a variant of `ChiSquareDist` with faster but less accurate methods. The non-static version of `inverseF` calls the static version. This method is not very accurate for small n but becomes better as n increases. The other methods are the same as in `ChiSquareDist`.

```
package umontreal.iro.lecuyer.probdist;
```

```
public class ChiSquareDistQuick extends ChiSquareDist
```

Constructor

```
public ChiSquareDistQuick (int n)
```

Constructs a chi-square distribution with `n` degrees of freedom.

Methods

```
public static double inverseF (int n, double u)
```

Computes a quick-and-dirty approximation of $F^{-1}(u)$, where F is the *chi-square* distribution with n degrees of freedom. Uses the approximation given in Figure L.24 of [7] over most of the range. For $u < 0.02$ or $u > 0.98$, it uses the approximation given in [22] for $n \geq 10$, and returns `2.0 * GammaDist.inverseF (n/2, 6, u)` for $n < 10$ in order to avoid the loss of precision of the above approximations. When $n \geq 10$ or $0.02 < u < 0.98$, it is between 20 to 30 times faster than the same method in `ChiSquareDist` for n between 10 and 1000 and even faster for larger n.

Note that the number d of decimal digits of precision generally increases with n. For $n = 3$, we only have $d = 3$ over most of the range. For $n = 10$, $d = 5$ except far in the tails where $d = 3$. For $n = 100$, one has more than $d = 7$ over most of the range and for $n = 1000$, at least $d = 8$. The cases $n = 1$ and $n = 2$ are exceptions, with precision of about $d = 10$.

ChiSquareNoncentralDist

Extends the class `ContinuousDistribution` for the *noncentral chi-square* distribution with ν degrees of freedom and noncentrality parameter λ, where $\nu > 0$ and $\lambda > 0$ [27, page 436]. Its density is

$$f(x) = \frac{e^{-(x+\lambda)/2}}{2} \left(\frac{x}{\lambda}\right)^{(\nu-2)/4} I_{\nu/2-1}\left(\sqrt{\lambda x}\right) \qquad \text{for } x > 0, \tag{37}$$

where $I_\nu(x)$ is the modified Bessel function of the first kind of order ν given by

$$I_\nu(z) = \sum_{j=0}^{\infty} \frac{(z/2)^{\nu+2j}}{j!\,\Gamma(\nu+j+1)}, \tag{38}$$

where $\Gamma(x)$ is the gamma function. Notice that this distribution is more general than the *chi-square* distribution since its number of degrees of freedom can be any positive real number. For $\lambda = 0$ and ν a positive integer, we have the ordinary *chi-square* distribution.

The cumulative probability function can be written as

$$P[X \le x] = \sum_{j=0}^{\infty} \frac{e^{-\lambda/2}(\lambda/2)^j}{j!} P[\chi^2_{\nu+2j} \le x], \tag{39}$$

where $\chi^2_{\nu+2j}$ is the *central chi-square* distribution with $\nu + 2j$ degrees of freedom.

```
package umontreal.iro.lecuyer.probdist;

public class ChiSquareNoncentralDist extends ContinuousDistribution
```

Constructor

```
public ChiSquareNoncentralDist (double nu, double lambda)
```

Constructs a noncentral chi-square distribution with $\nu =$ `nu` degrees of freedom and noncentrality parameter $\lambda =$ `lambda`.

Methods

```
public static double density (double nu, double lambda, double x)
```

Computes the density function (37) for a *noncentral chi-square* distribution with $\nu =$ `nu` degrees of freedom and parameter $\lambda =$ `lambda`.

```
public static double cdf (double nu, double lambda, double x)
```

Computes the noncentral chi-square distribution function (39) with $\nu =$ `nu` degrees of freedom and parameter $\lambda =$ `lambda`.

```
public static double barF (double nu, double lambda, double x)
```

Computes the complementary noncentral chi-square distribution function with $\nu =$ `nu` degrees of freedom and parameter $\lambda =$ `lambda`.

```
public static double inverseF (double nu, double lambda, double u)
```

Computes the inverse of the noncentral chi-square distribution with $\nu =$ `nu` degrees of freedom and parameter $\lambda =$ `lambda`.

```
public static double getMean (double nu, double lambda)
```

Computes and returns the mean $E[X] = \nu + \lambda$ of the noncentral chi-square distribution with parameters $\nu =$ `nu` and $\lambda =$ `lambda`.

```
public static double getVariance (double nu, double lambda)
```

Computes and returns the variance $\mathrm{Var}[X] = 2(\nu + 2\lambda)$ of the noncentral chi-square distribution with parameters $\nu =$ `nu` and $\lambda =$ `lambda`.

```
public static double getStandardDeviation (double nu, double lambda)
```

Computes and returns the standard deviation of the noncentral chi-square distribution with parameters $\nu =$ `nu` and $\lambda =$ `lambda`.

```
public double getNu()
```

Returns the parameter ν of this object.

```
public double getLambda()
```

Returns the parameter λ of this object.

```
public void setParams (double nu, double lambda)
```

Sets the parameters $\nu =$ `nu` and $\lambda =$ `lambda` of this object.

```
public double[] getParams ()
```

Returns a table containing the parameters of the current distribution.

ErlangDist

Extends the class `GammaDist` for the special case of the *Erlang* distribution with shape parameter $k > 0$ and scale parameter $\lambda > 0$. This distribution is a special case of the gamma distribution for which the shape parameter $k = \alpha$ is an integer.

```
package umontreal.iro.lecuyer.probdist;

public class ErlangDist extends GammaDist
```

Constructors

`public ErlangDist (int k)`

Constructs a `ErlangDist` object with parameters $k =$ `k` and $\lambda = 1$.

`public ErlangDist (int k, double lambda)`

Constructs a `ErlangDist` object with parameters $k =$ `k` and $\lambda =$ `lambda`.

Methods

`public static double density (int k, double lambda, double x)`

Computes the density function.

`public static double cdf (int k, double lambda, int d, double x)`

Computes the distribution function using the gamma distribution function.

`public static double barF (int k, double lambda, int d, double x)`

Computes the complementary distribution function.

`public static double inverseF (int k, double lambda, int d, double u)`

Returns the inverse distribution function.

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

Estimates the parameters (k, λ) of the Erlang distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[k, \lambda]$.

`public static ErlangDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of an Erlang distribution with parameters k and λ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (int k, double lambda)`

Computes and returns the mean, $E[X] = k/\lambda$, of the Erlang distribution with parameters k and λ.

`public static double getVariance (int k, double lambda)`

Computes and returns the variance, $\mathrm{Var}[X] = k/\lambda^2$, of the Erlang distribution with parameters k and λ.

`public static double getStandardDeviation (int k, double lambda)`

Computes and returns the standard deviation of the Erlang distribution with parameters k and λ.

`public int getK()`

Returns the parameter k for this object.

`public void setParams (int k, double lambda, int d)`

Sets the parameters k and λ of the distribution for this object. Non-static methods are computed with a rough target of `d` decimal digits of precision.

`public double[] getParams ()`

Return a table containing parameters of the current distribution. This table is put in regular order: $[k, \lambda]$.

ExponentialDist

Extends the class `ContinuousDistribution` for the *exponential* distribution [26, page 494] with mean $1/\lambda$ where $\lambda > 0$. Its density is

$$f(x) = \lambda e^{-\lambda x} \qquad \text{for } x \geq 0, \tag{40}$$

its distribution function is

$$F(x) = 1 - e^{-\lambda x}, \qquad \text{for } x \geq 0, \tag{41}$$

and its inverse distribution function is

$$F^{-1}(u) = -\ln(1-u)/\lambda, \qquad \text{for } 0 < u < 1.$$

```
package umontreal.iro.lecuyer.probdist;

public class ExponentialDist extends ContinuousDistribution
```

Constructors

```
public ExponentialDist()
```

Constructs an `ExponentialDist` object with parameter $\lambda = 1$.

```
public ExponentialDist (double lambda)
```

Constructs an `ExponentialDist` object with parameter $\lambda =$ `lambda`.

Methods

```
public static double density (double lambda, double x)
```

Computes the density function.

```
public static double cdf (double lambda, double x)
```

Computes the distribution function.

```
public static double barF (double lambda, double x)
```

Computes the complementary distribution function.

```
public static double inverseF (double lambda, double u)
```

Computes the inverse distribution function.

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

Estimates the parameter λ of the exponential distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimate is returned in a one-element array, as element 0.

`public static ExponentialDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of an exponential distribution with parameter λ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double lambda)`

Computes and returns the mean, $E[X] = 1/\lambda$, of the exponential distribution with parameter λ.

`public static double getVariance (double lambda)`

Computes and returns the variance, $\mathrm{Var}[X] = 1/\lambda^2$, of the exponential distribution with parameter λ.

`public static double getStandardDeviation (double lambda)`

Computes and returns the standard deviation of the exponential distribution with parameter λ.

`public double getLambda()`

Returns the value of λ for this object.

`public void setLambda (double lambda)`

Sets the value of λ for this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution.

ExponentialDistFromMean

Extends the `ExponentialDist` class with a constructor accepting as argument the mean $1/\lambda$ instead of the rate λ.

```
package umontreal.iro.lecuyer.probdist;

public class ExponentialDistFromMean extends ExponentialDist
```

Constructors

`public ExponentialDistFromMean (double mean)`

Constructs a new exponential distribution with mean `mean`.

Methods

`public void setMean (double mean)`

Calls `setLambda` with argument `1/mean` to change the mean of this distribution.

ExtremeValueDist

This class has been replaced by `GumbelDist`.

Extends the class `ContinuousDistribution` for the *extreme value* (or *Gumbel*) distribution [27, page 2], with location parameter α and scale parameter $\lambda > 0$. It has density

$$f(x) = \lambda e^{-\lambda(x-\alpha)} e^{-e^{-\lambda(x-\alpha)}}, \qquad \text{for } -\infty < x < \infty, \tag{42}$$

distribution function

$$F(x) = e^{-e^{-\lambda(x-\alpha)}} \qquad \text{for } -\infty < x < \infty, \tag{43}$$

and inverse distribution function

$$F^{-1}(u) = -\ln(-\ln(u))/\lambda + \alpha, \qquad \text{for } 0 \le u \le 1. \tag{44}$$

```
package umontreal.iro.lecuyer.probdist;

@Deprecated
public class ExtremeValueDist extends ContinuousDistribution
```

Constructors

`public ExtremeValueDist()`

THIS CLASS HAS BEEN REPLACED BY `GumbelDist`. Constructs a `ExtremeValueDist` object with parameters $\alpha = 0$ and $\lambda = 1$.

`public ExtremeValueDist (double alpha, double lambda)`

THIS CLASS HAS BEEN REPLACED BY `GumbelDist`. Constructs a `ExtremeValueDist` object with parameters α = `alpha` and λ = `lambda`.

Methods

`public static double density (double alpha, double lambda, double x)`

Computes the density function.

`public static double cdf (double alpha, double lambda, double x)`

THIS CLASS HAS BEEN REPLACED BY `GumbelDist`. Computes the distribution function.

`public static double barF (double alpha, double lambda, double x)`

Computes the complementary distribution function.

`public static double inverseF (double alpha, double lambda, double u)`

Computes the inverse distribution function.

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

Estimates the parameters (α, λ) of the extreme value distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\alpha,\ \lambda]$.

```
@Deprecated
public static double[] getMaximumLikelihoodEstimate (double[] x, int n)
```

Same as `getMLE`.

```
public static ExtremeValueDist getInstanceFromMLE (double[] x, int n)
```

Creates a new instance of an extreme value distribution with parameters α and λ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static double getMean (double alpha, double lambda)
```

Computes and returns the mean, $E[X] = \alpha + \gamma/\lambda$, of the extreme value distribution with parameters α and λ, where $\gamma = 0.5772156649$ is the Euler-Mascheroni constant.

```
public static double getVariance (double alpha, double lambda)
```

Computes and returns the variance, $\mathrm{Var}[X] = \pi^2/(6\lambda^2)$, of the extreme value distribution with parameters α and λ.

```
public static double getStandardDeviation (double alpha, double lambda)
```

Computes and returns the standard deviation of the extreme value distribution with parameters α and λ.

```
public double getAlpha()
```

Returns the parameter α of this object.

```
public double getLambda()
```

Returns the parameter λ of this object.

```
public void setParams (double alpha, double lambda)
```

Sets the parameters α and λ of this object.

```
public double[] getParams ()
```

Return a table containing the parameters of the current distribution. This table is put in regular order: $[\alpha,\ \lambda]$.

FatigueLifeDist

Extends the class `ContinuousDistribution` for the *fatigue life* distribution [4] with location parameter μ, scale parameter β and shape parameter γ. Its density is

$$f(x) = \left[\frac{\sqrt{\frac{x-\mu}{\beta}} + \sqrt{\frac{\beta}{x-\mu}}}{2\gamma(x-\mu)}\right] \phi\left(\frac{\sqrt{\frac{x-\mu}{\beta}} - \sqrt{\frac{\beta}{x-\mu}}}{\gamma}\right), \qquad \text{for } x > \mu, \tag{45}$$

where ϕ is the probability density of the standard normal distribution. The distribution function is given by

$$F(x) = \Phi\left(\frac{\sqrt{\frac{x-\mu}{\beta}} - \sqrt{\frac{\beta}{x-\mu}}}{\gamma}\right), \qquad \text{for } x > \mu, \tag{46}$$

where Φ is the standard normal distribution function. Restrictions: $\beta > 0$, $\gamma > 0$.

The non-static versions of the methods `cdf`, `barF`, and `inverseF` call the static version of the same name.

```
package umontreal.iro.lecuyer.probdist;

public class FatigueLifeDist extends ContinuousDistribution
```

Constructor

```
public FatigueLifeDist (double mu, double beta, double gamma)
```

Constructs a fatigue life distribution with parameters μ, β and γ.

Methods

```
public static double density (double mu, double beta, double gamma,
                              double x)
```

Computes the density (45) for the fatigue life distribution with parameters μ, β and γ.

```
public static double cdf (double mu, double beta, double gamma, double x)
```

Computes the fatigue life distribution function with parameters μ, β and γ.

```
public static double barF (double mu, double beta, double gamma,
                           double x)
```

Computes the complementary distribution function of the fatigue life distribution with parameters μ, β and γ.

```
public static double inverseF (double mu, double beta, double gamma,
                               double u)
```

Computes the inverse of the fatigue life distribution with parameters μ, β and γ.

```
public static double[] getMLE (double[] x, int n, double mu)
```

Estimates the parameters (μ, β, γ) of the fatigue life distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a three-element array, in regular order: [μ, β, γ].

```
public static double getMean (double mu, double beta, double gamma)
```

Computes and returns the mean $E[X] = \mu + \beta(1 + \gamma^2/2)$ of the fatigue life distribution with parameters μ, β and γ.

```
public static double getVariance (double mu, double beta, double gamma)
```

Computes and returns the variance $\mathrm{Var}[X] = \beta^2\gamma^2(1 + 5\gamma^2/4)$ of the fatigue life distribution with parameters μ, β and γ.

```
public static double getStandardDeviation (double mu, double beta,
                                           double gamma)
```

Computes and returns the standard deviation of the fatigue life distribution with parameters μ, β and γ.

```
public double getBeta()
```

Returns the parameter β of this object.

```
public double getGamma()
```

Returns the parameter γ of this object.

```
public double getMu()
```

Returns the parameter μ of this object.

```
public void setParams (double mu, double beta, double gamma)
```

Sets the parameters μ, β and γ of this object.

```
public double[] getParams ()
```

Return a table containing the parameters of the current distribution. This table is put in regular order: [μ, β, γ].

FisherFDist

Extends the class `ContinuousDistribution` for the *Fisher F* distribution with n_1 and n_2 degrees of freedom, where n_1 and n_2 are positive integers. Its density is

$$f(x) = \frac{\Gamma(\frac{n_1+n_2}{2}) n_1^{\frac{n_1}{2}} n_2^{\frac{n_2}{2}}}{\Gamma(\frac{n_1}{2})\Gamma(\frac{n_2}{2})} \frac{x^{\frac{n_1-2}{2}}}{(n_2 + n_1 x)^{\frac{n_1+n_2}{2}}}, \qquad \text{for } x > 0 \tag{47}$$

where $\Gamma(x)$ is the gamma function defined in (50).

```
package umontreal.iro.lecuyer.probdist;

public class FisherFDist extends ContinuousDistribution
```

Constructor

`public FisherFDist (int n1, int n2)`

Constructs a Fisher F distribution with `n1` and `n2` degrees of freedom.

Methods

`public static double density (int n1, int n2, double x)`

Computes the density function (47) for a Fisher F distribution with `n1` and `n2` degrees of freedom, evaluated at x.

`public static double cdf (int n1, int n2, double x)`

Computes the distribution function of the Fisher F distribution with parameters `n1` and `n2`, evaluated at x.

`public static double barF (int n1, int n2, double x)`

Computes the complementary distribution function of the Fisher F distribution with parameters `n1` and `n2`, evaluated at x.

`public static double inverseF (int n1, int n2, double u)`

Computes the inverse of the Fisher F distribution with parameters `n1` and `n2`, evaluated at u.

`public static double getMean (int n1, int n2)`

Computes and returns the mean $E[X] = n_2/(n_2 - 2)$ of the Fisher F distribution with parameters `n1` and `n2` $= n_2$.

`public static double getVariance (int n1, int n2)`

Computes and returns the variance

$$\mathrm{Var}[X] = \frac{2n_2^2(n_2 + n_1 - 2)}{n_1(n_2 - 2)^2(n_2 - 4)}$$

of the Fisher F distribution with parameters `n1` $= n_1$ and `n2` $= n_2$.

`public static double getStandardDeviation (int n1, int n2)`

Computes and returns the standard deviation of the Fisher F distribution with parameters `n1` and `n2`.

`public int getN1()`

Returns the parameter `n1` of this object.

`public int getN2()`

Returns the parameter `n2` of this object.

`public void setParams (int n1, int n2)`

Sets the parameters `n1` and `n2` of this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: [`n1`, `n2`].

FoldedNormalDist

Extends the class `ContinuousDistribution` for the *folded normal* distribution with parameters $\mu \geq 0$ and $\sigma > 0$. The density is

$$f(x) = \phi\left(\frac{x-\mu}{\sigma}\right) + \phi\left(\frac{-x-\mu}{\sigma}\right) \qquad \text{for } x \geq 0, \tag{48}$$

$$f(x) = 0, \qquad \text{for } x < 0,$$

where ϕ denotes the density function of a standard normal distribution.

```
package  umontreal.iro.lecuyer.probdist;

public class FoldedNormalDist extends ContinuousDistribution
```

Constructors

```
public FoldedNormalDist (double mu, double sigma)
```

Constructs a `FoldedNormalDist` object with parameters $\mu =$ `mu` and $\sigma =$ `sigma`.

Methods

```
public static double density (double mu, double sigma, double x)
```

Computes the density function of the *folded normal* distribution.

```
public static double cdf (double mu, double sigma, double x)
```

Computes the distribution function.

```
public static double barF (double mu, double sigma, double x)
```

Computes the complementary distribution function.

```
public static double inverseF (double mu, double sigma, double u)
```

Computes the inverse of the distribution function.

```
public static double getMean (double mu, double sigma)
```

Computes and returns the mean

$$E[X] = \sigma\sqrt{\frac{2}{\pi}}\, e^{-\mu^2/(2\sigma^2)} + \mu \operatorname{erf}\left(\frac{\mu}{\sigma\sqrt{2}}\right),$$

where $\operatorname{erf}(z)$ is the error function.

`public static double getVariance (double mu, double sigma)`

Computes and returns the variance

$$\mathrm{Var}[X] = \mu^2 + \sigma^2 - E[X]^2.$$

`public static double getStandardDeviation (double mu, double sigma)`

Computes the standard deviation of the folded normal distribution with parameters μ and σ.

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

NOT IMPLEMENTED. Les formules pour le MLE sont données dans [34].

`public double getMu()`

Returns the parameter μ of this object.

`public double getSigma()`

Returns the parameter σ of this object.

`public void setParams (double mu, double sigma)`

Sets the parameters μ and σ for this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: [μ, σ].

`public String toString ()`

Returns a `String` containing information about the current distribution.

FrechetDist

Extends the class `ContinuousDistribution` for the *Fréchet* distribution [27, page 3], with location parameter δ, scale parameter $\beta > 0$, and shape parameter $\alpha > 0$, where we use the notation $z = (x - \delta)/\beta$. It has density

$$f(x) = \frac{\alpha e^{-z^{-\alpha}}}{\beta z^{\alpha+1}}, \qquad \text{for } x > \delta$$

and distribution function

$$F(x) = e^{-z^{-\alpha}}, \qquad \text{for } x > \delta.$$

Both the density and the distribution are 0 for $x \le \delta$.

The mean is given by

$$E[X] = \delta + \beta\Gamma\left(1 - \frac{1}{\alpha}\right),$$

where $\Gamma(x)$ is the gamma function. The variance is

$$\mathrm{Var}[X] = \beta^2 \left[\Gamma\left(1 - \frac{2}{\alpha}\right) - \Gamma^2\left(1 - \frac{1}{\alpha}\right)\right].$$

```
package umontreal.iro.lecuyer.probdist;

public class FrechetDist extends ContinuousDistribution
```

Constructors

```
public FrechetDist (double alpha)
```

Constructor for the standard *Fréchet* distribution with parameters $\beta = 1$ and $\delta = 0$.

```
public FrechetDist (double alpha, double beta, double delta)
```

Constructs a `FrechetDist` object with parameters $\alpha =$ `alpha`, $\beta =$ `beta` and $\delta =$ `delta`.

Methods

```
public static double density (double alpha, double beta, double delta,
                              double x)
```

Computes and returns the density function.

```
public static double cdf (double alpha, double beta, double delta,
                          double x)
```

Computes and returns the distribution function.

```
public static double barF (double alpha, double beta, double delta,
                           double x)
```

Computes and returns the complementary distribution function $1 - F(x)$.

```
public static double inverseF (double alpha, double beta, double delta,
                               double u)
```

Computes and returns the inverse distribution function.

```
public static double[] getMLE (double[] x, int n, double delta)
```

Given $\delta =$ `delta`, estimates the parameters (α, β) of the *Fréchet* distribution using the maximum likelihood method with the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: [α, β].

```
public static FrechetDist getInstanceFromMLE (double[] x, int n,
                                              double delta)
```

Given $\delta =$ `delta`, creates a new instance of a *Fréchet* distribution with parameters α and β estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static double getMean (double alpha, double beta, double delta)
```

Returns the mean of the *Fréchet* distribution with parameters α, β and δ.

```
public static double getVariance (double alpha, double beta,
                                  double delta)
```

Returns the variance of the *Fréchet* distribution with parameters α, β and δ.

```
public static double getStandardDeviation (double alpha, double beta,
                                           double delta)
```

Returns the standard deviation of the *Fréchet* distribution with parameters α, β and δ.

```
public double getAlpha()
```

Returns the parameter α of this object.

```
public double getBeta()
```

Returns the parameter β of this object.

```
public double getDelta()
```

Returns the parameter δ of this object.

```
public void setParams (double alpha, double beta, double delta)
```

Sets the parameters α, β and δ of this object.

```
public double[] getParams()
```

Return an array containing the parameters of the current object in regular order: [α, β, δ].

GammaDist

Extends the class `ContinuousDistribution` for the *gamma* distribution [26, page 337] with shape parameter $\alpha > 0$ and scale parameter $\lambda > 0$. The density is

$$f(x) = \frac{\lambda^\alpha x^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)}, \qquad \text{for } x > 0, \tag{49}$$

where Γ is the gamma function, defined by

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx. \tag{50}$$

In particular, $\Gamma(n) = (n-1)!$ when n is a positive integer.

```
package umontreal.iro.lecuyer.probdist;

public class GammaDist extends ContinuousDistribution
```

Constructors

`public GammaDist (double alpha)`

Constructs a `GammaDist` object with parameters α = `alpha` and $\lambda = 1$.

`public GammaDist (double alpha, double lambda)`

Constructs a `GammaDist` object with parameters α = `alpha` and λ = `lambda`.

`public GammaDist (double alpha, double lambda, int d)`

Constructs a `GammaDist` object with parameters α = `alpha` and λ = `lambda`, and approximations of roughly `d` decimal digits of precision when computing functions.

Methods

`public static double density (double alpha, double lambda, double x)`

Computes the density function (49) at x.

`public static double cdf (double alpha, double lambda, int d, double x)`

Returns an approximation of the gamma distribution function with parameters α = `alpha` and λ = `lambda`, whose density is given by (49). The approximation is an improved version of the algorithm in [3]. The function tries to return d decimals digits of precision. For α not too large (e.g., $\alpha \le 1000$), d gives a good idea of the precision attained.

`public static double cdf (double alpha, int d, double x)`

Equivalent to `cdf (alpha, 1.0, d, x)`.

`public static double barF (double alpha, double lambda, int d, double x)`

Computes the complementary distribution function.

`public static double barF (double alpha, int d, double x)`

Same as `barF (alpha, 1.0, d, x)`.

```
public static double inverseF (double alpha, double lambda, int d,
                               double u)
```

Computes the inverse distribution function.

`public static double inverseF (double alpha, int d, double u)`

Same as `inverseF (alpha, 1, d, u)`.

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

Estimates the parameters (α, λ) of the gamma distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\alpha, \lambda]$.

`public static GammaDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a gamma distribution with parameters α and λ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double alpha, double lambda)`

Computes and returns the mean $E[X] = \alpha/\lambda$ of the gamma distribution with parameters α and λ.

`public static double getVariance (double alpha, double lambda)`

Computes and returns the variance $\mathrm{Var}[X] = \alpha/\lambda^2$ of the gamma distribution with parameters α and λ.

`public static double getStandardDeviation (double alpha, double lambda)`

Computes and returns the standard deviation of the gamma distribution with parameters α and λ.

`public double getAlpha()`

Return the parameter α for this object.

`public double getLambda()`

Return the parameter λ for this object.

`public void setParams (double alpha, double lambda, int d)`

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[\alpha, \lambda]$.

GammaDistFromMoments

Extends the `GammaDist` distribution with constructors accepting the mean μ and variance σ^2 as arguments instead of a shape parameter α and a scale parameter λ. Since $\mu = \alpha/\lambda$, and $\sigma^2 = \alpha/\lambda^2$, the shape and scale parameters are $\alpha = \mu^2/\sigma^2$, and $\lambda = \mu/\sigma^2$, respectively.

```
package umontreal.iro.lecuyer.probdist;

public class GammaDistFromMoments extends GammaDist
```

Constructors

```
public GammaDistFromMoments (double mean, double var, int d)
```

Constructs a gamma distribution with mean `mean`, variance `var`, and `d` decimal of precision.

```
public GammaDistFromMoments (double mean, double var)
```

Constructs a gamma distribution with mean `mean`, and variance `var`.

GumbelDist

Extends the class `ContinuousDistribution` for the *Gumbel* distribution [27, page 2], with location parameter δ and scale parameter $\beta \neq 0$. Using the notation $z = (x - \delta)/\beta$, it has density

$$f(x) = \frac{e^{-z} e^{-e^{-z}}}{|\beta|}, \qquad \text{for } -\infty < x < \infty \tag{51}$$

and distribution function

$$F(x) = \begin{cases} e^{-e^{-z}}, & \text{for } \beta > 0 \\ 1 - e^{-e^{-z}}, & \text{for } \beta < 0. \end{cases}$$

```
package umontreal.iro.lecuyer.probdist;

public class GumbelDist extends ContinuousDistribution
```

Constructors

```
public GumbelDist()
```

Constructor for the standard Gumbel distribution with parameters $\beta = 1$ and $\delta = 0$.

```
public GumbelDist (double beta, double delta)
```

Constructs a `GumbelDist` object with parameters β = `beta` and δ = `delta`.

Methods

```
public static double density (double beta, double delta, double x)
```

Computes and returns the density function.

```
public static double cdf (double beta, double delta, double x)
```

Computes and returns the distribution function.

```
public static double barF (double beta, double delta, double x)
```

Computes and returns the complementary distribution function $1 - F(x)$.

```
public static double inverseF (double beta, double delta, double u)
```

Computes and returns the inverse distribution function.

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

Estimates the parameters (β, δ) of the Gumbel distribution, *assuming that* $\beta > 0$, and using the maximum likelihood method with the n observations $x[i]$, $i = 0, 1, \ldots, n - 1$. The estimates are returned in a two-element array, in regular order: $[\beta, \delta]$.

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

Similar to `getMLE`, but *for the case* $\beta < 0$.

`public static GumbelDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of an Gumbel distribution with parameters β and δ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$, *assuming that* $\beta > 0$.

`public static GumbelDist getInstanceFromMLEmin (double[] x, int n)`

Similar to `getInstanceFromMLE`, but *for the case* $\beta < 0$.

`public static double getMean (double beta, double delta)`

Returns the mean, $E[X] = \delta + \gamma\beta$, of the Gumbel distribution with parameters β and δ, where $\gamma = 0.5772156649015329$ is the Euler-Mascheroni constant.

`public static double getVariance (double beta, double delta)`

Returns the variance $\text{Var}[X] = \pi^2\beta^2/6$ of the Gumbel distribution with parameters β and δ.

`public static double getStandardDeviation (double beta, double delta)`

Returns the standard deviation of the Gumbel distribution with parameters β and δ.

`public double getBeta()`

Returns the parameter β of this object.

`public double getDelta()`

Returns the parameter δ of this object.

`public void setParams (double beta, double delta)`

Sets the parameters β and δ of this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: [β, δ].

HalfNormalDist

Extends the class `ContinuousDistribution` for the *half-normal* distribution with parameters μ and $\sigma > 0$. Its density is

$$f(x) = \frac{1}{\sigma}\sqrt{\frac{2}{\pi}}\, e^{-(x-\mu)^2/2\sigma^2}, \qquad \text{for } x \ge \mu. \tag{52}$$

$$f(x) = 0, \qquad \text{for } x < \mu.$$

```
package  umontreal.iro.lecuyer.probdist;

public class HalfNormalDist extends ContinuousDistribution
```

Constructors

`public HalfNormalDist (double mu, double sigma)`

Constructs a `HalfNormalDist` object with parameters $\mu =$ `mu` and $\sigma =$ `sigma`.

Methods

`public static double density (double mu, double sigma, double x)`

Computes the density function of the *half-normal* distribution.

`public static double cdf (double mu, double sigma, double x)`

Computes the distribution function.

`public static double barF (double mu, double sigma, double x)`

Computes the complementary distribution function.

`public static double inverseF (double mu, double sigma, double u)`

Computes the inverse of the distribution function.

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

Estimates the parameters μ and σ of the half-normal distribution using the maximum likelihood method from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array: $[\mu, \sigma]$.

`public static double[] getMLE (double[] x, int n, double mu)`

Estimates the parameter σ of the half-normal distribution using the maximum likelihood method from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$ and the parameter $\mu =$ `mu`. The estimate is returned in a one-element array: $[\sigma]$.

`public static double getMean (double mu, double sigma)`

Computes and returns the mean $E[X] = \mu + \sigma\sqrt{2/\pi}$.

`public static double getVariance (double mu, double sigma)`

Computes and returns the variance $\mathrm{Var}[X] = (1 - 2/\pi)\,\sigma^2$.

`public static double getStandardDeviation (double mu, double sigma)`

Computes the standard deviation of the half-normal distribution with parameters μ and σ.

`public double getMu()`

Returns the parameter μ of this object.

`public double getSigma()`

Returns the parameter σ of this object.

`public void setParams (double mu, double sigma)`

Sets the parameters μ and σ.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: [μ, σ].

`public String toString ()`

Returns a `String` containing information about the current distribution.

HyperbolicSecantDist

Extends the class `ContinuousDistribution` for the *hyperbolic secant* distribution with location parameter μ and scale parameter $\sigma > 0$. Its density is

$$f(x) = \frac{1}{2\sigma} \operatorname{sech}\left(\frac{\pi}{2}\frac{(x-\mu)}{\sigma}\right) \tag{53}$$

The distribution function is given by

$$F(x) = \frac{2}{\pi} \tan^{-1}\left[\exp\left(\frac{\pi}{2}\frac{(x-\mu)}{\sigma}\right)\right] \tag{54}$$

The non-static versions of the methods `cdf`, `barF`, and `inverseF` call the static version of the same name.

```
package umontreal.iro.lecuyer.probdist;

public class HyperbolicSecantDist extends ContinuousDistribution
```

Constructor

`public HyperbolicSecantDist (double mu, double sigma)`

Constructs a hyperbolic secant distribution with parameters μ and σ.

Methods

`public static double density (double mu, double sigma, double x)`

Computes the density function (53) for a hyperbolic secant distribution with parameters μ and σ.

`public static double cdf (double mu, double sigma, double x)`

Computes the distribution function of the hyperbolic secant distribution with parameters μ and σ.

`public static double barF (double mu, double sigma, double x)`

Computes the complementary distribution function of the hyperbolic secant distribution with parameters μ and σ.

`public static double inverseF (double mu, double sigma, double u)`

Computes the inverse of the hyperbolic secant distribution with parameters μ and σ.

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

Estimates the parameters (μ, σ) of the hyperbolic secant distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: [μ, σ].

`public static HyperbolicSecantDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a hyperbolic secant distribution with parameters μ and σ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n - 1$.

`public static double getMean (double mu, double sigma)`

Computes and returns the mean $E[X] = \mu$ of the hyperbolic secant distribution with parameters μ and σ.

`public static double getVariance (double mu, double sigma)`

Computes and returns the variance $\mathrm{Var}[X] = \sigma^2$ of the hyperbolic secant distribution with parameters μ and σ.

`public static double getStandardDeviation (double mu, double sigma)`

Computes and returns the standard deviation of the hyperbolic secant distribution with parameters μ and σ.

`public double getMu()`

Returns the parameter μ of this object.

`public double getSigma()`

Returns the parameter σ of this object.

`public void setParams (double mu, double sigma)`

Sets the parameters μ and σ of this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: [μ, σ].

HypoExponentialDist

This class implements the *hypoexponential* distribution, also called the generalized Erlang distribution. Let the X_j, $j = 1, \dots, k$, be k independent exponential random variables with different rates λ_j, i.e. assume that $\lambda_j \neq \lambda_i$ for $i \neq j$. Then the sum $\sum_{j=1}^{k} X_j$ is called a *hypoexponential* random variable.

Let the $k \times k$ upper triangular bidiagonal matrix

$$\mathbf{A} = \begin{pmatrix} -\lambda_1 & \lambda_1 & 0 & \dots & 0 \\ 0 & -\lambda_2 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -\lambda_{k-1} & \lambda_{k-1} \\ 0 & \dots & 0 & 0 & -\lambda_k \end{pmatrix} \tag{55}$$

with λ_j the rates of the k exponential random variables; then the cumulative complementary probability of the hypoexponential distribution is given by [43, 32]

$$\bar{F}(x) = \mathbb{P}\left[X_1 + \cdots + X_k > x\right] = \sum_{j=1}^{k} \left(e^{\mathbf{A}x}\right)_{1j}, \tag{56}$$

i.e., it is the sum of the elements of the first row of matrix $e^{\mathbf{A}x}$. The density of the hypoexponential distribution is

$$f(x) = \left(-e^{\mathbf{A}x}\mathbf{A}\right)_{1k} = \lambda_k \left(e^{\mathbf{A}x}\right)_{1k}, \tag{57}$$

i.e., it is element $(1, k)$ of matrix $-e^{\mathbf{A}x}\mathbf{A}$. The distribution function is as usual $F(x) = 1 - \bar{F}(x)$.

See the class `HypoExponentialDistQuick` for alternative formulae for the probabilities.

```
package umontreal.iro.lecuyer.probdist;


public class HypoExponentialDist extends ContinuousDistribution
```

Constructor

`public HypoExponentialDist (double[] lambda)`

Constructs a `HypoExponentialDist` object, with rates $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

Methods

`public static double density (double[] lambda, double x)`

Computes the density function $f(x)$, with $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

`public static double cdf (double[] lambda, double x)`

Computes the distribution function $F(x)$, with $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

`public static double barF (double[] lambda, double x)`

Computes the complementary distribution $\bar{F}(x)$, with $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

`public static double inverseF (double[] lambda, double u)`

Computes the inverse distribution function $F^{-1}(u)$, with $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$. It uses a root-finding method and is very slow.

`public static double getMean (double[] lambda)`

Returns the mean, $E[X] = \sum_{i=1}^{k} 1/\lambda_i$, of the hypoexponential distribution with rates $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

`public static double getVariance (double[] lambda)`

Returns the variance, $\mathrm{Var}[X] = \sum_{i=1}^{k} 1/\lambda_i^2$, of the hypoexponential distribution with rates $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

`public static double getStandardDeviation (double[] lambda)`

Returns the standard deviation of the hypoexponential distribution with rates $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

`public double[] getLambda()`

Returns the value of λ_i for this object.

`public void setLambda (double[] lambda)`

Sets the value of $\lambda_i =$`lambda`$[i-1]$, $i = 1, \dots, k$ for this object.

`public double[] getParams()`

Same as `getLambda`.

HypoExponentialDistQuick

This class is a subclass of `HypoExponentialDist` and also implements the *hypoexponential* distribution. It uses different algorithms to compute the probabilities. The formula (56) for the complementary distribution is mathematically equivalent to (see [50, page 299] and [21, Appendix B])

$$\bar{F}(x) = \mathbb{P}\left[X_1 + \cdots + X_k > x\right] = \sum_{i=1}^{k} e^{-\lambda_i x} \prod_{\substack{j=1\\ j\neq i}}^{k} \frac{\lambda_j}{\lambda_j - \lambda_i}. \tag{58}$$

The expression (58) is much faster to compute than the matrix exponential formula (56), but it becomes numerically unstable when k gets large and/or the differences between the λ_i are too small, because it is an alternating sum with relatively large terms of similar size. When the λ_i are close, many of the factors $\lambda_j - \lambda_i$ in (58) are small, and the effect of this is amplified when k is large. This gives rise to large terms of opposite sign in the sum and the formula becomes unstable due to subtractive cancellation. For example, with the computations done in standard 64-bit floating-point arithmetic, if the λ_i are regularly spaced with differences of $\lambda_{i+1} - \lambda_i = 0.1$ for all i, the formula (58) breaks down already for $k \approx 15$, while if the differences $\lambda_{i+1} - \lambda_i = 3$, it gives a few decimal digits of precision for k up to ≈ 300.

The formula (57) for the density is mathematically equivalent to the much faster formula

$$f(x) = \sum_{i=1}^{k} \lambda_i e^{-\lambda_i x} \prod_{\substack{j=1\\ j\neq i}}^{k} \frac{\lambda_j}{\lambda_j - \lambda_i}, \tag{59}$$

which is also numerically unstable when k gets large and/or the differences between the λ_i are too small.

```
package umontreal.iro.lecuyer.probdist;

public class HypoExponentialDistQuick extends HypoExponentialDist
```

Constructor

`public HypoExponentialDistQuick (double[] lambda)`

Constructs a `HypoExponentialDistQuick` object, with rates λ_i = `lambda[`$i-1$`]`, $i = 1, \ldots, k$.

Methods

`public static double density (double[] lambda, double x)`

Computes the density function $f(x)$, with $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

`public static double cdf (double[] lambda, double x)`

Computes the distribution function $F(x)$, with $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

`public static double barF (double[] lambda, double x)`

Computes the complementary distribution $\bar{F}(x)$, with $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$.

`public static double inverseF (double[] lambda, double u)`

Computes the inverse distribution function $F^{-1}(u)$, with $\lambda_i =$ `lambda`$[i-1]$, $i = 1, \dots, k$. It uses a root-finding method and is very slow.

InverseGammaDist

Extends the class `ContinuousDistribution` for the *inverse gamma* distribution with shape parameter $\alpha > 0$ and scale parameter $\beta > 0$. The density function is given by

$$f(x) = \begin{cases} \dfrac{\beta^\alpha e^{-\beta/x}}{x^{\alpha+1}\Gamma(\alpha)} & \text{for } x > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{60}$$

where Γ is the gamma function. The distribution function is given by

$$F(x) = 1 - F_G\left(\frac{1}{x}\right) \qquad \text{for } x > 0, \tag{61}$$

and $F(x) = 0$ otherwise, where $F_G(x)$ is the distribution function of a gamma distribution with shape parameter α and scale parameter β.

```
package umontreal.iro.lecuyer.probdist;

public class InverseGammaDist extends ContinuousDistribution
```

Constructor

`public InverseGammaDist (double alpha, double beta)`

Constructs an `InverseGammaDist` object with parameters α = `alpha` and β = `beta`.

Methods

`public static double density (double alpha, double beta, double x)`

Computes the density function of the inverse gamma distribution with shape parameter α and scale parameter β.

`public static double cdf (double alpha, double beta, double x)`

Computes the cumulative probability function of the inverse gamma distribution with shape parameter α and scale parameter β.

`public static double barF (double alpha, double beta, double x)`

Computes the complementary distribution function of the inverse gamma distribution with shape parameter α and scale parameter β.

`public static double inverseF (double alpha, double beta, double u)`

Computes the inverse distribution function of the inverse gamma distribution with shape parameter α and scale parameter β.

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

Estimates the parameters (α, β) of the inverse gamma distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\alpha, \beta]$.

`public static InverseGammaDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of the inverse gamma distribution with parameters α and β estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double alpha, double beta)`

Returns the mean $E[X] = \beta/(\alpha-1)$ of the inverse gamma distribution with shape parameter α and scale parameter β.

`public static double getVariance (double alpha, double beta)`

Returns the variance $\mathrm{Var}[X] = \beta^2/((\alpha-1)^2(\alpha-2))$ of the inverse gamma distribution with shape parameter α and scale parameter β.

`public static double getStandardDeviation (double alpha, double beta)`

Returns the standard deviation of the inverse gamma distribution with shape parameter α and scale parameter β.

`public double getAlpha()`

Returns the α parameter of this object.

`public double getBeta()`

Returns the β parameter of this object.

`public void setParam (double alpha, double beta)`

Sets the parameters α and β of this object.

`public double[] getParams ()`

Returns a table containing the parameters of the current distribution. This table is put in regular order: $[\alpha, \beta]$.

InverseGaussianDist

Extends the class `ContinuousDistribution` for the *inverse Gaussian* distribution with location parameter $\mu > 0$ and scale parameter $\lambda > 0$. Its density is

$$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}}\, e^{-\lambda(x-\mu)^2/(2\mu^2 x)}, \qquad \text{for } x > 0. \tag{62}$$

The distribution function is given by

$$F(x) = \Phi\left(\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} - 1\right)\right) + e^{(2\lambda/\mu)}\Phi\left(-\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} + 1\right)\right), \tag{63}$$

where Φ is the standard normal distribution function.

The non-static versions of the methods `cdf`, `barF`, and `inverseF` call the static version of the same name.

```
package umontreal.iro.lecuyer.probdist;

public class InverseGaussianDist extends ContinuousDistribution
```

Constructor

`public InverseGaussianDist (double mu, double lambda)`

Constructs the *inverse Gaussian* distribution with parameters μ and λ.

Methods

`public static double density (double mu, double lambda, double x)`

Computes the density function (62) for the inverse gaussian distribution with parameters μ and λ, evaluated at x.

`public static double cdf (double mu, double lambda, double x)`

Computes the distribution function (63) of the inverse gaussian distribution with parameters μ and λ, evaluated at x.

`public static double barF (double mu, double lambda, double x)`

Computes the complementary distribution function of the inverse gaussian distribution with parameters μ and λ, evaluated at x.

`public static double inverseF (double mu, double lambda, double u)`

Computes the inverse of the inverse gaussian distribution with parameters μ and λ.

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

Estimates the parameters (μ, λ) of the inverse gaussian distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\mu, \lambda]$.

`public static InverseGaussianDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of an inverse gaussian distribution with parameters μ and λ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double mu, double lambda)`

Returns the mean $E[X] = \mu$ of the inverse gaussian distribution with parameters μ and λ.

`public static double getVariance (double mu, double lambda)`

Computes and returns the variance $\mathrm{Var}[X] = \mu^3/\lambda$ of the inverse gaussian distribution with parameters μ and λ.

`public static double getStandardDeviation (double mu, double lambda)`

Computes and returns the standard deviation of the inverse gaussian distribution with parameters μ and λ.

`public double getLambda()`

Returns the parameter λ of this object.

`public double getMu()`

Returns the parameter μ of this object.

`public void setParams (double mu, double lambda)`

Sets the parameters μ and λ of this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[\mu, \lambda]$.

JohnsonSBDist

Extends the class `ContinuousDistribution` for the *Johnson* S_B distribution (see [33, page 314]) with shape parameters γ and $\delta > 0$, location parameter ξ, and scale parameter $\lambda > 0$. Denoting $y = (x - \xi)/\lambda$, the density is

$$f(x) = \frac{\delta}{\lambda y(1-y)\sqrt{2\pi}} \exp(-(1/2)\left[\gamma + \delta \ln\left(y/(1-y)\right)\right]^2) \text{ for } \xi < x < \xi + \lambda, \tag{64}$$

and 0 elsewhere. The distribution function is

$$F(x) = \Phi[\gamma + \delta \ln(y/(1-y))], \text{ for } \xi < x < \xi + \lambda, \tag{65}$$

where Φ is the standard normal distribution function. The inverse distribution function is

$$F^{-1}(u) = \xi + \lambda(1/(1 + e^{-v(u)})) \qquad \text{for } 0 \le u \le 1, \tag{66}$$

where

$$v(u) = [\Phi^{-1}(u) - \gamma]/\delta. \tag{67}$$

This class relies on the methods `NormalDist.cdf01` and `NormalDist.inverseF01` of `NormalDist` to approximate Φ and Φ^{-1}.

```
package umontreal.iro.lecuyer.probdist;

public class JohnsonSBDist extends ContinuousDistribution
```

Constructor

```
public JohnsonSBDist (double gamma, double delta,
                      double xi, double lambda)
```

Constructs a `JohnsonSBDist` object with shape parameters γ and δ, location parameter ξ and scale parameter λ.

Methods

```
public static double density (double gamma, double delta,
                              double xi, double lambda, double x)
```

Computes the density function (64).

```
public static double cdf (double gamma, double delta,
                          double xi, double lambda, double x)
```

Computes the distribution function (65).

```
public static double barF (double gamma, double delta,
                           double xi, double lambda, double x)
```

Computes the complementary distribution.

```
public static double inverseF (double gamma, double delta,
                               double xi, double lambda, double u)
```

Computes the inverse of the distribution (66).

```
public double getGamma()
```

Returns the value of γ for this object.

```
public double getDelta()
```

Returns the value of δ for this object.

```
public double getXi()
```

Returns the value of ξ for this object.

```
public double getLambda()
```

Returns the value of λ for this object.

```
public void setParams(double gamma, double delta, double xi,
                      double lambda)
```

Sets the value of the parameters γ, δ, ξ and λ for this object.

```
public double[] getParams ()
```

Return a table containing the parameters of the current distribution. This table is put in regular order: [γ, δ, ξ, λ].

JohnsonSUDist

Extends the class `ContinuousDistribution` for the *Johnson* S_U distribution (see [33, page 316]). It has shape parameters γ and $\delta > 0$, location parameter ξ, and scale parameter $\lambda > 0$. Denoting $y = (x - \xi)/\lambda$, the distribution has density

$$f(x) = \frac{\delta}{\lambda\sqrt{y^2+1}\sqrt{2\pi}} \exp\left(-(1/2)\left[\gamma + \delta \ln\left[y + \sqrt{y^2+1}\right]\right]^2\right) \qquad \text{for } -\infty < x < \infty, \tag{68}$$

and distribution function

$$F(x) = \Phi\left\{\gamma + \delta \ln\left[y + \sqrt{y^2+1}\right]\right\}, \qquad \text{for } -\infty < x < \infty, \tag{69}$$

where Φ is the standard normal distribution function. The inverse distribution function is

$$F^{-1}(u) = \xi + \lambda(e^{t(u)} - e^{-t(u)})/2, \qquad \text{for } 0 \le u < 1, \tag{70}$$

where

$$t(u) = [\Phi^{-1}(u) - \gamma]/\delta. \tag{71}$$

This class relies on the methods `NormalDist.cdf01` and `NormalDist.inverseF01` of `NormalDist` to approximate Φ and Φ^{-1}.

```
package umontreal.iro.lecuyer.probdist;

public class JohnsonSUDist extends ContinuousDistribution
```

Constructors

```
public JohnsonSUDist (double gamma, double delta)
```

Same as `JohnsonSUDist` (gamma, delta, 0.0, 1.0).

```
public JohnsonSUDist (double gamma, double delta,
                      double xi, double lambda)
```

Constructs a `JohnsonSUDist` object with shape parameters γ and δ, location parameter ξ, and scale parameter λ.

Methods

```
public static double density (double gamma, double delta,
                              double xi, double lambda, double x)
```

Computes the density function $f(x)$.

```
public static double cdf (double gamma, double delta,
                          double xi, double lambda, double x)
```

Computes the distribution function $F(x)$.

```
public static double barF (double gamma, double delta,
                           double xi, double lambda, double x)
```

Computes the complementary distribution function $1 - F(x)$.

```
public static double inverseF (double gamma, double delta,
                               double xi, double lambda, double u)
```

Computes the inverse distribution function $F^{-1}(u)$.

```
public static double getMean (double gamma, double delta,
                              double xi, double lambda)
```

Computes and returns the mean $E[X] = \xi - \lambda e^{1/(2\delta^2)} \sinh(\frac{\gamma}{\delta})$ of the Johnson S_U distribution with parameters γ, δ, ξ and λ.

```
public static double getVariance (double gamma, double delta,
                                  double xi, double lambda)
```

Computes and returns the variance $\mathrm{Var}[X] = \lambda^2[(e^{1/\delta^2} - 1)(e^{1/\delta^2}\cosh(2\frac{\gamma}{\delta}) + 1)]/2$ of the Johnson S_U distribution with parameters γ, δ, ξ and λ.

```
public static double getStandardDeviation (double gamma, double delta,
                                           double xi, double lambda)
```

Computes and returns the standard deviation of the Johnson S_U distribution with parameters γ, δ, ξ and λ.

```
public double getGamma()
```

Returns the value of γ for this object.

```
public double getDelta()
```

Returns the value of δ for this object.

```
public double getXi()
```

Returns the value of ξ for this object.

```
public double getLambda()
```

Returns the value of λ for this object.

```
public void setParams (double gamma, double delta,
                       double xi, double lambda)
```

Sets the value of the parameters γ, δ, ξ and λ for this object.

```
public double[] getParams ()
```

Return a table containing the parameters of the current distribution. This table is put in regular order: [γ, δ, ξ, λ].

LaplaceDist

Extends the class `ContinuousDistribution` for the *Laplace* distribution (see, e.g., [27, page 165]). It has location parameter μ and scale parameter $\beta > 0$. The density function is given by

$$f(x) = \frac{e^{-|x-\mu|/\beta}}{2\beta} \qquad \text{for } -\infty < x < \infty. \tag{72}$$

The distribution function is

$$F(x) = \begin{cases} \frac{1}{2} e^{(x-\mu)/\beta} & \text{if } x \le \mu, \\ 1 - \frac{1}{2} e^{(\mu-x)/\beta} & \text{otherwise,} \end{cases} \tag{73}$$

and its inverse is

$$F^{-1}(u) = \begin{cases} \beta \log(2u) + \mu & \text{if } 0 \le u \le \frac{1}{2}, \\ \mu - \beta \log(2(1-u)) & \text{otherwise.} \end{cases} \tag{74}$$

```
package umontreal.iro.lecuyer.probdist;

public class LaplaceDist extends ContinuousDistribution
```

Constructors

```
public LaplaceDist()
```

Constructs a `LaplaceDist` object with default parameters $\mu = 0$ and $\beta = 1$.

```
public LaplaceDist (double mu, double beta)
```

Constructs a `LaplaceDist` object with parameters $\mu =$ `mu` and $\beta =$ `beta`.

Methods

```
public static double density (double mu, double beta, double x)
```

Computes the Laplace density function.

```
public static double cdf (double mu, double beta, double x)
```

Computes the Laplace distribution function.

```
public static double barF (double mu, double beta, double x)
```

Computes the Laplace complementary distribution function.

```
public static double inverseF (double mu, double beta, double u)
```

Computes the inverse Laplace distribution function.

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

Estimates the parameters (μ, β) of the Laplace distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\mu,\ \beta]$.

`public static LaplaceDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a Laplace distribution with parameters μ and β estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double mu, double beta)`

Computes and returns the mean $E[X] = \mu$ of the Laplace distribution with parameters μ and β.

`public static double getVariance (double mu, double beta)`

Computes and returns the variance $\mathrm{Var}[X] = 2\beta^2$ of the Laplace distribution with parameters μ and β.

`public static double getStandardDeviation (double mu, double beta)`

Computes and returns the standard deviation of the Laplace distribution with parameters μ and β.

`public double getMu()`

Returns the parameter μ.

`public double getBeta()`

Returns the parameter β.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[\mu,\ \beta]$.

LogisticDist

Extends the class `ContinuousDistribution` for the *logistic* distribution (e.g., [27, page 115]). It has location parameter α and scale parameter $\lambda > 0$. The density is

$$f(x) = \frac{\lambda e^{-\lambda(x-\alpha)}}{(1 + e^{-\lambda(x-\alpha)})^2} \qquad \text{for } -\infty < x < \infty, \tag{75}$$

and the distribution function is

$$F(x) = \frac{1}{1 + e^{-\lambda(x-\alpha)}} \qquad \text{for } -\infty < x < \infty. \tag{76}$$

For $\lambda = 1$ and $\alpha = 0$, one can write

$$F(x) = \frac{1 + \tanh(x/2)}{2}. \tag{77}$$

The inverse distribution function is given by

$$F^{-1}(u) = \ln(u/(1-u))/\lambda + \alpha \qquad \text{for } 0 \le u < 1.$$

```
package umontreal.iro.lecuyer.probdist;

public class LogisticDist extends ContinuousDistribution
```

Constructors

`public LogisticDist()`

Constructs a `LogisticDist` object with default parameters $\alpha = 0$ and $\lambda = 1$.

`public LogisticDist (double alpha, double lambda)`

Constructs a `LogisticDist` object with parameters α = `alpha` and λ = `lambda`.

Methods

`public static double density (double alpha, double lambda, double x)`

Computes the density function $f(x)$.

`public static double cdf (double alpha, double lambda, double x)`

Computes the distribution function $F(x)$.

`public static double barF (double alpha, double lambda, double x)`

Computes the complementary distribution function $1 - F(x)$.

`public static double inverseF (double alpha, double lambda, double u)`

Computes the inverse distribution function $F^{-1}(u)$.

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

Estimates the parameters (α, λ) of the logistic distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\alpha, \lambda]$.

`public static LogisticDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a logistic distribution with parameters α and λ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double alpha, double lambda)`

Computes and returns the mean $E[X] = \alpha$ of the logistic distribution with parameters α and λ.

`public static double getVariance (double alpha, double lambda)`

Computes and returns the variance $\mathrm{Var}[X] = \pi^2/(3\lambda^2)$ of the logistic distribution with parameters α and λ.

`public static double getStandardDeviation (double alpha, double lambda)`

Computes and returns the standard deviation of the logistic distribution with parameters α and λ.

`public double getAlpha()`

Return the parameter α of this object.

`public double getLambda()`

Returns the parameter λ of this object.

`public void setParams (double alpha, double lambda)`

Sets the parameters α and λ of this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[\alpha, \lambda]$.

LoglogisticDist

Extends the class `ContinuousDistribution` for the *Log-Logistic* distribution with shape parameter $\alpha > 0$ and scale parameter $\beta > 0$. Its density is

$$f(x) = \frac{\alpha (x/\beta)^{\alpha-1}}{\beta [1 + (x/\beta)^\alpha]^2} \qquad \text{for } x > 0 \tag{78}$$

and its distribution function is

$$F(x) = \frac{1}{1 + (\frac{x}{\beta})^{-\alpha}} \qquad \text{for } x > 0. \tag{79}$$

The complementary distribution is

$$\bar{F}(x) = \frac{1}{1 + (\frac{x}{\beta})^{\alpha}} \qquad \text{for } x > 0. \tag{80}$$

```
package umontreal.iro.lecuyer.probdist;

public class LoglogisticDist extends ContinuousDistribution
```

Constructor

```
public LoglogisticDist (double alpha, double beta)
```

Constructs a log-logistic distribution with parameters α and β.

Methods

```
public static double density (double alpha, double beta, double x)
```

Computes the density function (78) for a log-logisitic distribution with parameters α and β.

```
public static double cdf (double alpha, double beta, double x)
```

Computes the distribution function (79) of the log-logistic distribution with parameters α and β.

```
public static double barF (double alpha, double beta, double x)
```

Computes the complementary distribution function (80) of the log-logistic distribution with parameters α and β.

```
public static double inverseF (double alpha, double beta, double u)
```

Computes the inverse of the log-logistic distribution with parameters α and β.

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

Estimates the parameters (α, β) of the log-logistic distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: [α, β].

`public static LoglogisticDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a log-logistic distribution with parameters α and β estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double alpha, double beta)`

Computes and returns the mean $E[X] = \beta\theta \operatorname{cosec}(\theta)$, where $\theta = \pi/\alpha$, of the log-logistic distribution with parameters α and β.

`public static double getVariance (double alpha, double beta)`

Computes and returns the variance $\mathrm{Var}[X] = \beta^2\theta(2\mathrm{cosec}(2\theta) - \theta[\mathrm{cosec}(\theta)]^2)$, where $\theta = \pi/\alpha$, of the log-logistic distribution with parameters α and β.

`public static double getStandardDeviation (double alpha, double beta)`

Computes and returns the standard deviation of the log-logistic distribution with parameters α and β.

`public double getAlpha()`

Return the parameter α of this object.

`public double getBeta()`

Returns the parameter β of this object.

`public void setParams (double alpha, double beta)`

Sets the parameters α and β of this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: [α, β].

LognormalDist

Extends the class `ContinuousDistribution` for the *lognormal* distribution [26]. It has scale parameter μ and shape parameter $\sigma > 0$. The density is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\ln(x)-\mu)^2/(2\sigma^2)} \qquad \text{for } x > 0, \tag{81}$$

and 0 elsewhere. The distribution function is

$$F(x) = \Phi\left((\ln(x) - \mu)/\sigma\right) \qquad \text{for } x > 0, \tag{82}$$

where Φ is the standard normal distribution function. Its inverse is given by

$$F^{-1}(u) = e^{\mu + \sigma\Phi^{-1}(u)} \qquad \text{for } 0 \le u < 1. \tag{83}$$

If $\ln(Y)$ has a *normal* distribution, then Y has a *lognormal* distribution with the same parameters.

This class relies on the methods `NormalDist.cdf01` and `NormalDist.inverseF01` of `NormalDist` to approximate Φ and Φ^{-1}.

```
package umontreal.iro.lecuyer.probdist;

public class LognormalDist extends ContinuousDistribution
```

Constructors

`public LognormalDist()`

Constructs a `LognormalDist` object with default parameters $\mu = 0$ and $\sigma = 1$.

`public LognormalDist (double mu, double sigma)`

Constructs a `LognormalDist` object with parameters $\mu =$ `mu` and $\sigma =$ `sigma`.

Methods

`public static double density (double mu, double sigma, double x)`

Computes the lognormal density function $f(x)$ in (81).

`public static double cdf (double mu, double sigma, double x)`

Computes the lognormal distribution function, using `cdf01`.

`public static double barF (double mu, double sigma, double x)`

Computes the lognormal complementary distribution function $\bar{F}(x)$, using `NormalDist.barF01`.

```
public static double inverseF (double mu, double sigma, double u)
```

Computes the inverse of the lognormal distribution function, using `NormalDist.inverseF01`.

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

Estimates the parameters (μ, σ) of the lognormal distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\mu, \sigma]$.

```
public static LognormalDist getInstanceFromMLE (double[] x, int n)
```

Creates a new instance of a lognormal distribution with parameters μ and σ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static double getMean (double mu, double sigma)
```

Computes and returns the mean $E[X] = e^{\mu+\sigma^2/2}$ of the lognormal distribution with parameters μ and σ.

```
public static double getVariance (double mu, double sigma)
```

Computes and returns the variance $\mathrm{Var}[X] = e^{2\mu+\sigma^2}(e^{\sigma^2} - 1)$ of the lognormal distribution with parameters μ and σ.

```
public static double getStandardDeviation (double mu, double sigma)
```

Computes and returns the standard deviation of the lognormal distribution with parameters μ and σ.

```
public double getMu()
```

Returns the parameter μ of this object.

```
public double getSigma()
```

Returns the parameter σ of this object.

```
public void setParams (double mu, double sigma)
```

Sets the parameters μ and σ of this object.

```
public double[] getParams ()
```

Return a table containing the parameters of the current distribution. This table is put in regular order: $[\mu, \sigma]$.

LognormalDistFromMoments

Extends the `LognormalDist` class with a constructor accepting the mean m and the variance v of the distribution as arguments. The mean and variance of a lognormal random variable with parameters μ and σ are $e^{\mu+\sigma^2/2}$ and $e^{2\mu+\sigma^2}(e^{\sigma^2}-1)$ respectively, so the parameters are given by $\sigma = \sqrt{\ln(v/m^2+1)}$ and $\mu = \ln(m) - \sigma^2/2$.

```
package umontreal.iro.lecuyer.probdist;

public class LognormalDistFromMoments extends LognormalDist
```

Constructor

```
public LognormalDistFromMoments (double mean, double var)
```

NakagamiDist

Extends the class `ContinuousDistribution` for the *Nakagami* distribution with location parameter a, scale parameter $\lambda > 0$ and shape parameter $c > 0$. The density is

$$f(x) = \frac{2\lambda^c}{\Gamma(c)}(x-a)^{2c-1}e^{-\lambda(x-a)^2} \qquad \text{for } x > a, \tag{84}$$

$$f(x) = 0 \qquad \text{for } x \le a,$$

where Γ is the gamma function.

```
package  umontreal.iro.lecuyer.probdist;
```

```
public class NakagamiDist extends ContinuousDistribution
```

Constructors

```
public NakagamiDist (double a, double lambda, double c)
```

Constructs a `NakagamiDist` object with parameters $a =$ `a`, $\lambda =$ `lambda` and $c =$ `c`.

Methods

```
public static double density (double a, double lambda, double c,
                              double x)
```

Computes the density function of the *Nakagami* distribution.

```
public static double cdf (double a, double lambda, double c, double x)
```

Computes the distribution function.

```
public static double barF (double a, double lambda, double c, double x)
```

Computes the complementary distribution function.

```
public static double inverseF (double a, double lambda, double c,
                               double u)
```

Computes the inverse of the distribution function.

```
public static double getMean (double a, double lambda, double c)
```

Computes and returns the mean

$$E[X] = a + \frac{1}{\sqrt{\lambda}}\,\frac{\Gamma(c+1/2)}{\Gamma(c)}.$$

```
public static double getVariance (double a, double lambda, double c)
```

Computes and returns the variance

$$\mathrm{Var}[X] = \frac{1}{\lambda}\left[c - \left(\frac{\Gamma(c+1/2)}{\Gamma(c)}\right)^2\right].$$

```
public static double getStandardDeviation (double a, double lambda,
                                           double c)
```

Computes the standard deviation of the Nakagami distribution with parameters a, λ and c.

```
public double getA()
```

Returns the location parameter a of this object.

```
public double getLambda()
```

Returns the scale parameter λ of this object.

```
public double getC()
```

Returns the shape parameter c of this object.

```
public void setParams (double a, double lambda, double c)
```

Sets the parameters a, λ and c of this object.

```
public double[] getParams ()
```

Return a table containing the parameters of the current distribution. This table is put in regular order: [a, λ, c].

```
public String toString ()
```

Returns a `String` containing information about the current distribution.

NormalDist

Extends the class `ContinuousDistribution` for the *normal* distribution (e.g., [26, page 80]). It has mean μ and variance σ^2. Its density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)} \qquad \text{for } -\infty < x < \infty, \tag{85}$$

where $\sigma > 0$. When $\mu = 0$ and $\sigma = 1$, we have the *standard normal* distribution, with corresponding distribution function

$$F(x) = \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt \qquad \text{for } -\infty < x < \infty. \tag{86}$$

The non-static methods `cdf`, `barF`, and `inverseF` are implemented via `cdf01`, `barF01`, and `inverseF01`, respectively.

```
package umontreal.iro.lecuyer.probdist;

public class NormalDist extends ContinuousDistribution
```

Constructors

`public NormalDist()`

Constructs a `NormalDist` object with default parameters $\mu = 0$ and $\sigma = 1$.

`public NormalDist (double mu, double sigma)`

Constructs a `NormalDist` object with mean $\mu =$ `mu` and standard deviation $\sigma =$ `sigma`.

Methods

`public static double density01 (double x)`

Same as `density (0, 1, x)`.

`public static double density (double mu, double sigma, double x)`

Computes the normal density function (85).

`public static double cdf01 (double x)`

Same as `cdf (0, 1, x)`.

`public static double cdf (double mu, double sigma, double x)`

Computes the normal distribution function with mean μ and variance σ^2. Uses the Chebyshev approximation proposed in [52], which gives 16 decimals of precision.

`public static double barF01 (double x)`

Same as `barF (0, 1, x)`.

`public static double barF (double mu, double sigma, double x)`

Computes the complementary normal distribution function $\bar{F}(x) = 1 - \Phi((x - \mu)/\sigma)$, with mean μ and variance σ^2. Uses a Chebyshev series giving 16 decimal digits of precision [52].

`public static double inverseF01 (double u)`

Same as `inverseF (0, 1, u)`.

`public static double inverseF (double mu, double sigma, double u)`

Computes the inverse normal distribution function with mean μ and variance σ^2. Uses different rational Chebyshev approximations [5]. Returns 16 decimal digits of precision for $2.2 \times 10^{-308} < u < 1$.

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

Estimates the parameters (μ, σ) of the normal distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n - 1$. The estimates are returned in a two-element array, in regular order: $[\hat{\mu}, \hat{\sigma}]$.

`public static NormalDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a normal distribution with parameters μ and σ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n - 1$.

`public static double getMean (double mu, double sigma)`

Computes and returns the mean $E[X] = \mu$ of the normal distribution with parameters μ and σ.

`public static double getVariance (double mu, double sigma)`

Computes and returns the variance $\mathrm{Var}[X] = \sigma^2$ of the normal distribution with parameters μ and σ.

`public static double getStandardDeviation (double mu, double sigma)`

Computes and returns the standard deviation σ of the normal distribution with parameters μ and σ.

`public double getMu()`

Returns the parameter μ.

`public double getSigma()`

Returns the parameter σ.

`public void setParams (double mu, double sigma)`

Sets the parameters μ and σ of this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[\mu, \sigma]$.

NormalDistQuick

A variant of the class `NormalDist` (for the *normal* distribution with mean μ and variance σ^2). The difference is in the implementation of the methods `cdf01`, `barF01` and `inverseF01`, which are faster but less accurate than those of the class `NormalDist`.

```
package umontreal.iro.lecuyer.probdist;

public class NormalDistQuick extends NormalDist
```

Constructors

`public NormalDistQuick()`

Constructs a `NormalDistQuick` object with default parameters $\mu = 0$ and $\sigma = 1$.

`public NormalDistQuick (double mu, double sigma)`

Constructs a `NormalDistQuick` object with mean $\mu =$ `mu` and standard deviation $\sigma =$ `sigma`.

Methods

`public static double cdf01 (double x)`

Same as `cdf (0.0, 1.0, x)`.

`public static double cdf (double mu, double sigma, double x)`

Returns an approximation of $\Phi(x)$, where Φ is the standard normal distribution function, with mean 0 and variance 1. Uses Marsaglia et al's [40] fast method with table lookups. Returns 15 decimal digits of precision. This method is approximately 60% faster than `NormalDist.cdf`.

`public static double barF01 (double x)`

Same as `barF (0.0, 1.0, x)`.

`public static double barF (double mu, double sigma, double x)`

Returns an approximation of $1 - \Phi(x)$, where Φ is the standard normal distribution function, with mean 0 and variance 1. Uses Marsaglia et al's [40] fast method with table lookups. Returns 15 decimal digits of precision. This method is approximately twice faster than `NormalDist.barF`.

`public static double inverseF01 (double u)`

Same as `inverseF (0.0, 1.0, u)`.

`public static double inverseF (double mu, double sigma, double u)`

Returns an approximation of $\Phi^{-1}(u)$, where Φ is the standard normal distribution function, with mean 0 and variance 1. Uses the method of Marsaglia, Zaman, and Marsaglia [40], with table lookups. Returns 6 decimal digits of precision. This method is approximately 20% faster than `NormalDist.inverseF`.

NormalInverseGaussianDist

Extends the class `ContinuousDistribution` for the *normal inverse gaussian* distribution with location parameter μ, scale parameter $\delta > 0$, tail heavyness $\alpha > 0$, and asymmetry parameter β such that $0 \le |\beta| < \alpha$. Its density is

$$f(x) = \frac{\alpha\delta e^{\delta\gamma+\beta(x-\mu)} K_1\left(\alpha\sqrt{\delta^2+(x-\mu)^2}\right)}{\pi\sqrt{\delta^2+(x-\mu)^2}}, \qquad \text{for } -\infty < x < \infty, \tag{87}$$

where K_1 is the modified Bessel function of the second kind of order 1, and $\gamma = \sqrt{\alpha^2-\beta^2}$.

The distribution function is given by

$$F(x) = \int_{-\infty}^{x} dt f(t), \tag{88}$$

```
package umontreal.iro.lecuyer.probdist;

public class NormalInverseGaussianDist extends ContinuousDistribution
```

Constructor

```
public NormalInverseGaussianDist (double alpha, double beta, double mu,
                                  double delta)
```

Constructor for a *normal inverse gaussian* distribution with parameters α = `alpha`, β = `beta`, μ = `mu` and δ = `delta`.

Methods

```
public static double density (double alpha, double beta, double mu,
                              double delta, double x)
```

Computes the density function (87) for the *normal inverse gaussian* distribution with parameters α, β, μ and δ, evaluated at x.

```
public static double cdf (double alpha, double beta, double mu,
                          double delta, double x)
```

NOT IMPLEMENTED. Computes the distribution function (88) of the *normal inverse gaussian* distribution with parameters α, β, μ and δ, evaluated at x.

```
public static double barF (double alpha, double beta, double mu,
                           double delta, double x)
```

NOT IMPLEMENTED. Computes the complementary distribution function of the *normal inverse gaussian* distribution with parameters α, β, μ and δ, evaluated at x.

```
public static double inverseF (double alpha, double beta, double mu,
                               double delta, double u)
```

NOT IMPLEMENTED. Computes the inverse of the *normal inverse gaussian* distribution with parameters α, β, μ and δ.

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

NOT IMPLEMENTED.

```
public static NormalInverseGaussianDist getInstanceFromMLE (double[] x,
                                                            int n)
```

NOT IMPLEMENTED.

```
public static double getMean (double alpha, double beta, double mu,
                              double delta)
```

Returns the mean $E[X] = \mu + \delta\beta/\gamma$ of the *normal inverse gaussian* distribution with parameters α, β, μ and δ.

```
public static double getVariance (double alpha, double beta, double mu,
                                  double delta)
```

Computes and returns the variance $\mathrm{Var}[X] = \delta\alpha^2/\gamma^3$ of the *normal inverse gaussian* distribution with parameters α, β, μ and δ.

```
public static double getStandardDeviation (double alpha, double beta,
                                           double mu, double delta)
```

Computes and returns the standard deviation of the *normal inverse gaussian* distribution with parameters α, β, μ and δ.

```
public double getAlpha()
```

Returns the parameter α of this object.

```
public double getBeta()
```

Returns the parameter β of this object.

```
public double getMu()
```

Returns the parameter μ of this object.

```
public double getDelta()
```

Returns the parameter δ of this object.

```
public void setParams (double alpha, double beta, double mu,
                       double delta)
```

Sets the parameters α, β, μ and δ of this object.

```
public double[] getParams ()
```

Returns a table containing the parameters of the current distribution. This table is put in regular order: [α, β, μ, δ].

ParetoDist

Extends the class `ContinuousDistribution` for a distribution from the *Pareto* family, with shape parameter $\alpha > 0$ and location parameter $\beta > 0$ [26, page 574]. The density for this type of Pareto distribution is

$$f(x) = \frac{\alpha\beta^\alpha}{x^{\alpha+1}} \qquad \text{for } x \geq \beta, \tag{89}$$

and 0 otherwise. The distribution function is

$$F(x) = 1 - (\beta/x)^\alpha \qquad \text{for } x \geq \beta, \tag{90}$$

and the inverse distribution function is

$$F^{-1}(u) = \beta(1-u)^{-1/\alpha} \qquad \text{for } 0 \leq u < 1.$$

```
package umontreal.iro.lecuyer.probdist;

public class ParetoDist extends ContinuousDistribution
```

Constructors

`public ParetoDist (double alpha)`

Constructs a `ParetoDist` object with parameters $\alpha =$ `alpha` and $\beta = 1$.

`public ParetoDist (double alpha, double beta)`

Constructs a `ParetoDist` object with parameters $\alpha =$ `alpha` and $\beta =$ `beta`.

Methods

`public static double density (double alpha, double beta, double x)`

Computes the density function.

`public static double cdf (double alpha, double beta, double x)`

Computes the distribution function.

`public static double barF (double alpha, double beta, double x)`

Computes the complementary distribution function.

`public static double inverseF (double alpha, double beta, double u)`

Computes the inverse of the distribution function.

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

Estimates the parameters (α, β) of the Pareto distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: [α, β].

`public static ParetoDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a Pareto distribution with parameters α and β estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

`public static double getMean (double alpha, double beta)`

Computes and returns the mean $E[X] = \alpha\beta/(\alpha - 1)$ of the Pareto distribution with parameters α and β.

`public static double getVariance (double alpha, double beta)`

Computes and returns the variance $\mathrm{Var}[X] = \frac{\alpha\beta^2}{(\alpha-2)(\alpha-1)}$ of the Pareto distribution with parameters α and β.

`public static double getStandardDeviation (double alpha, double beta)`

Computes and returns the standard deviation of the Pareto distribution with parameters α and β.

`public double getAlpha()`

Returns the parameter α.

`public double getBeta()`

Returns the parameter β.

`public void setParams (double alpha, double beta)`

Sets the parameter α and β for this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: [α, β].

Pearson5Dist

THIS CLASS HAS BEEN RENAMED `InverseGammaDist`.

Extends the class `ContinuousDistribution` for the *Pearson type V* distribution with shape parameter $\alpha > 0$ and scale parameter $\beta > 0$. The density function is given by

$$f(x) = \begin{cases} \dfrac{\beta^\alpha e^{-\beta/x}}{x^{\alpha+1}\Gamma(\alpha)} & \text{for } x > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{91}$$

where Γ is the gamma function. The distribution function is given by

$$F(x) = 1 - F_G\left(\frac{1}{x}\right) \qquad \text{for } x > 0, \tag{92}$$

and $F(x) = 0$ otherwise, where $F_G(x)$ is the distribution function of a gamma distribution with shape parameter α and scale parameter β.

```
package umontreal.iro.lecuyer.probdist;

@Deprecated
public class Pearson5Dist extends ContinuousDistribution
```

Constructor

`public Pearson5Dist (double alpha, double beta)`

THIS CLASS HAS BEEN RENAMED `InverseGammaDist`. Constructs a `Pearson5Dist` object with parameters α = `alpha` and β = `beta`.

Methods

`public static double density (double alpha, double beta, double x)`

Computes the density function of a Pearson V distribution with shape parameter α and scale parameter β.

`public static double cdf (double alpha, double beta, double x)`

Computes the density function of a Pearson V distribution with shape parameter α and scale parameter β.

`public static double barF (double alpha, double beta, double x)`

Computes the complementary distribution function of a Pearson V distribution with shape parameter α and scale parameter β.

```
public static double inverseF (double alpha, double beta, double u)
```

Computes the inverse distribution function of a Pearson V distribution with shape parameter α and scale parameter β.

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

Estimates the parameters (α, β) of the Pearson V distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: [α, β].

```
public static Pearson5Dist getInstanceFromMLE (double[] x, int n)
```

Creates a new instance of a Pearson V distribution with parameters α and β estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static double getMean (double alpha, double beta)
```

Computes and returns the mean $E[X] = \beta/(\alpha - 1)$ of a Pearson V distribution with shape parameter α and scale parameter β.

```
public static double getVariance (double alpha, double beta)
```

Computes and returns the variance $\mathrm{Var}[X] = \beta^2/((\alpha-1)^2(\alpha-2)$ of a Pearson V distribution with shape parameter α and scale parameter β.

```
public static double getStandardDeviation (double alpha, double beta)
```

Computes and returns the standard deviation of a Pearson V distribution with shape parameter α and scale parameter β.

```
public double getAlpha()
```

Returns the α parameter of this object.

```
public double getBeta()
```

Returns the β parameter of this object.

```
public void setParam (double alpha, double beta)
```

Sets the parameters α and β of this object.

```
public double[] getParams ()
```

Return a table containing the parameters of the current distribution. This table is put in regular order: [α, β].

Pearson6Dist

Extends the class `ContinuousDistribution` for the *Pearson type VI* distribution with shape parameters $\alpha_1 > 0$ and $\alpha_2 > 0$, and scale parameter $\beta > 0$. The density function is given by

$$f(x) = \begin{cases} \dfrac{(x/\beta)^{\alpha_1 - 1}}{\beta \mathcal{B}(\alpha_1, \alpha_2)(1 + x/\beta)^{\alpha_1 + \alpha_2}} & \text{for } x > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{93}$$

where $\mathcal{B}$ is the beta function. The distribution function is given by

$$F(x) = F_B\left(\frac{x}{x + \beta}\right) \qquad \text{for } x > 0, \tag{94}$$

and $F(x) = 0$ otherwise, where $F_B(x)$ is the distribution function of a beta distribution with shape parameters α_1 and α_2.

```
package umontreal.iro.lecuyer.probdist;

public class Pearson6Dist extends ContinuousDistribution
```

Constructor

```
public Pearson6Dist (double alpha1, double alpha2, double beta)
```

Constructs a `Pearson6Dist` object with parameters α_1 = `alpha1`, α_2 = `alpha2` and β = `beta`.

Methods

```
public static double density (double alpha1, double alpha2,
                              double beta, double x)
```

Computes the density function of a Pearson VI distribution with shape parameters α_1 and α_2, and scale parameter β.

```
public static double cdf (double alpha1, double alpha2,
                          double beta, double x)
```

Computes the distribution function of a Pearson VI distribution with shape parameters α_1 and α_2, and scale parameter β.

```
public static double barF (double alpha1, double alpha2,
                           double beta, double x)
```

Computes the complementary distribution function of a Pearson VI distribution with shape parameters α_1 and α_2, and scale parameter β.

```
public static double inverseF (double alpha1, double alpha2,
                               double beta, double u)
```

Computes the inverse distribution function of a Pearson VI distribution with shape parameters α_1 and α_2, and scale parameter β.

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

Estimates the parameters $(\alpha_1, \alpha_2, \beta)$ of the Pearson VI distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a three-element array, in regular order: $[\alpha_1, \alpha_2, \beta]$.

```
public static Pearson6Dist getInstanceFromMLE (double[] x, int n)
```

Creates a new instance of a Pearson VI distribution with parameters α_1, α_2 and β, estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static double getMean (double alpha1, double alpha2,
                              double beta)
```

Computes and returns the mean $E[X] = (\beta\alpha_1)/(\alpha_2 - 1)$ of a Pearson VI distribution with shape parameters α_1 and α_2, and scale parameter β.

```
public static double getVariance (double alpha1, double alpha2,
                                  double beta)
```

Computes and returns the variance $\mathrm{Var}[X] = [\beta^2\alpha_1(\alpha_1 + \alpha_2 - 1)]/[(\alpha_2 - 1)^2(\alpha_2 - 2)]$ of a Pearson VI distribution with shape parameters α_1 and α_2, and scale parameter β.

```
public static double getStandardDeviation (double alpha1, double alpha2,
                                           double beta)
```

Computes and returns the standard deviation of a Pearson VI distribution with shape parameters α_1 and α_2, and scale parameter β.

```
public double getAlpha1()
```

Returns the α_1 parameter of this object.

```
public double getAlpha2()
```

Returns the α_2 parameter of this object.

```
public double getBeta()
```

Returns the β parameter of this object.

```
public void setParam (double alpha1, double alpha2, double beta)
```

Sets the parameters α_1, α_2 and β of this object.

```
public double[] getParams ()
```

Return a table containing the parameters of the current distribution. This table is put in regular order: $[\alpha_1, \alpha_2, \beta]$.

PiecewiseLinearEmpiricalDist

Extends the class `ContinuousDistribution` for a piecewise-linear approximation of the *empirical* distribution function, based on the observations $X_{(1)}, \dots, X_{(n)}$ (sorted by increasing order), and defined as follows (e.g., [33, page 318]). The distribution function starts at $X_{(1)}$ and climbs linearly by $1/(n-1)$ between any two successive observations. The density is

$$f(x) = \frac{1}{(n-1)(X_{(i+1)} - X_{(i)})} \text{ for } X_{(i)} \le x < X_{(i+1)} \text{ and } i = 1, 2, \dots, n-1. \tag{95}$$

The distribution function is

$$F(x) = \begin{cases} 0 & \text{for } x < X_{(1)}, \\ \dfrac{i-1}{n-1} + \dfrac{x - X_{(i)}}{(n-1)(X_{(i+1)} - X_{(i)})} & \text{for } X_{(i)} \le x < X_{(i+1)} \text{ and } i < n, \\ 1 & \text{for } x \ge X_{(n)}, \end{cases} \tag{96}$$

whose inverse is

$$F^{-1}(u) = X_{(i)} + ((n-1)u - i + 1)(X_{(i+1)} - X_{(i)}) \tag{97}$$

for $(i-1)/(n-1) \le u \le i/(n-1)$ and $i = 1, \dots, n-1$.

```
package umontreal.iro.lecuyer.probdist;

public class PiecewiseLinearEmpiricalDist extends ContinuousDistribution
```

```
public PiecewiseLinearEmpiricalDist (double[] obs)
```

Constructs a new piecewise-linear distribution using all the observations stored in `obs`. These observations are copied into an internal array and then sorted.

```
public PiecewiseLinearEmpiricalDist (Reader in) throws IOException
```

Constructs a new empirical distribution using the observations read from the reader `in`. This constructor will read the first `double` of each line in the stream. Any line that does not start with a `+`, `-`, or a decimal digit, is ignored. The file is read until its end. One must be careful about lines starting with a blank. This format is the same as in UNURAN.

```
public int getN()
```

Returns n, the number of observations.

```
public double getObs (int i)
```

Returns the value of $X_{(i)}$.

```
public double getSampleMean()
```

Returns the sample mean of the observations.

`public double getSampleVariance()`

Returns the sample variance of the observations.

`public double getSampleStandardDeviation()`

Returns the sample standard deviation of the observations.

`public double[] getParams ()`

Return a table containing parameters of the current distribution.

`public String toString ()`

Returns a `String` containing information about the current distribution.

PowerDist

Extends the class `ContinuousDistribution` for the *power* distribution [16, page 161] with shape parameter $c > 0$, over the interval $[a, b]$, where $a < b$. This distribution has density

$$f(x) = \frac{c(x-a)^{c-1}}{(b-a)^c}, \qquad \text{for } a \le x \le b, \tag{98}$$

and $f(x) = 0$ elsewhere. Its distribution function is

$$F(x) = \frac{(x-a)^c}{(b-a)^c}, \qquad \text{for } a \le x \le b, \tag{99}$$

with $F(x) = 0$ for $x \le a$ and $F(x) = 1$ for $x \ge b$.

```
package umontreal.iro.lecuyer.probdist;

public class PowerDist extends ContinuousDistribution
```

Constructors

`public PowerDist (double a, double b, double c)`

Constructs a `PowerDist` object with parameters $a =$ `a`, $b =$ `b` and $c =$ `c`.

`public PowerDist (double b, double c)`

Constructs a `PowerDist` object with parameters $a = 0$, $b =$ `b` and $c =$ `c`.

`public PowerDist (double c)`

Constructs a `PowerDist` object with parameters $a = 0$, $b = 1$ and $c =$ `c`.

Methods

`public static double density (double a, double b, double c, double x)`

Computes the density function (98).

`public static double cdf (double a, double b, double c, double x)`

Computes the distribution function (99).

`public static double barF (double a, double b, double c, double x)`

Computes the complementary distribution function.

`public static double inverseF (double a, double b, double c, double u)`

Computes the inverse of the distribution function.

`public static double[] getMLE (double[] x, int n, double a, double b)`

Estimates the parameter c of the power distribution from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$, using the maximum likelihood method and assuming that a and b are known. The estimate is returned in a one-element array: $[c]$.

```
public static PowerDist getInstanceFromMLE (double[] x, int n,
                                            double a, double b)
```

Creates a new instance of a power distribution with parameters a and b, with c estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, \ldots, n-1$.

`public static double getMean (double a, double b, double c)`

Returns the mean $a + (b-a)c/(c+1)$ of the power distribution with parameters a, b and c.

`public static double getVariance (double a, double b, double c)`

Computes and returns the variance $(b-a)^2 c/[(c+1)^2(c+2)]$ of the power distribution with parameters a, b and c.

`public static double getStandardDeviation (double a, double b, double c)`

Computes and returns the standard deviation of the power distribution with parameters a, b and c.

`public double getA()`

Returns the parameter a.

`public double getB()`

Returns the parameter b.

`public double getC()`

Returns the parameter c.

`public void setParams (double a, double b, double c)`

Sets the parameters a, b and c for this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[a, b, c]$.

RayleighDist

This class extends the class `ContinuousDistribution` for the *Rayleigh* distribution [16] with location parameter a, and scale parameter $\beta > 0$. The density function is

$$f(x) = \frac{(x-a)}{\beta^2} e^{-(x-a)^2/(2\beta^2)} \qquad \text{for } x \ge a, \tag{100}$$

and $f(x) = 0$ for $x < a$. The distribution function is

$$F(x) = 1 - e^{-(x-a)^2/(2\beta^2)} \qquad \text{for } x \ge a, \tag{101}$$

and the inverse distribution function is

$$F^{-1}(u) = x = a + \beta\sqrt{-2\ln(1-u)} \qquad \text{for } 0 \le u \le 1. \tag{102}$$

```
package umontreal.iro.lecuyer.probdist;

public class RayleighDist extends ContinuousDistribution
```

Constructors

`public RayleighDist (double beta)`

Constructs a `RayleighDist` object with parameters $a = 0$ and $\beta =$ `beta`.

`public RayleighDist (double a, double beta)`

Constructs a `RayleighDist` object with parameters $a =$ `a`, and $\beta =$ `beta`.

Methods

`public static double density (double a, double beta, double x)`

Computes the density function (100).

`public static double density (double beta, double x)`

Same as `density (0, beta, x)`.

`public static double cdf (double a, double beta, double x)`

Computes the distribution function (101).

`public static double cdf (double beta, double x)`

Same as `cdf (0, beta, x)`.

`public static double barF (double a, double beta, double x)`

Computes the complementary distribution function.

`public static double barF (double beta, double x)`

Same as `barF (0, beta, x)`.

`public static double inverseF (double a, double beta, double u)`

Computes the inverse of the distribution function (102).

`public static double inverseF (double beta, double u)`

Same as `inverseF (0, beta, u)`.

`public static double[] getMLE (double[] x, int n, double a)`

Estimates the parameter β of the Rayleigh distribution using the maximum likelihood method, assuming that a is known, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimate is returned in a one-element array: $[\hat{\beta}]$.

```
public static RayleighDist getInstanceFromMLE (double[] x, int n,
                                               double a)
```

Creates a new instance of a Rayleigh distribution with parameters a and $\hat{\beta}$. This last is estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, \ldots, n-1$.

`public static double getMean (double a, double beta)`

Returns the mean $a + \beta\sqrt{\pi/2}$ of the Rayleigh distribution with parameters a and β.

`public static double getVariance (double beta)`

Returns the variance of the Rayleigh distribution with parameter β.

`public static double getStandardDeviation (double beta)`

Returns the standard deviation $\beta\sqrt{2 - \pi/2}$ of the Rayleigh distribution with parameter β.

`public double getA()`

Returns the parameter a.

`public double getSigma()`

Returns the parameter β.

`public void setParams (double a, double beta)`

Sets the parameters a and β for this object.

`public double[] getParams ()`

Return an array containing the parameters of the current distribution in the order: $[a, \beta]$.

StudentDist

Extends the class `ContinuousDistribution` for the *Student* t-distribution [27, page 362] with n degrees of freedom, where n is a positive integer. Its density is

$$f(x) = \frac{\Gamma\left((n+1)/2\right)}{\Gamma(n/2)\sqrt{\pi n}} \left(1 + \frac{x^2}{n}\right)^{-(n+1)/2} \qquad \text{for } -\infty < x < \infty, \tag{103}$$

where $\Gamma(x)$ is the gamma function defined in (50).

```
package umontreal.iro.lecuyer.probdist;
```

```
public class StudentDist extends ContinuousDistribution
```

Constructors

```
public StudentDist (int n)
```

Constructs a `StudentDist` object with `n` degrees of freedom.

Methods

```
public static double density (int n, double x)
```

Computes the density function (103) of a Student t-distribution with n degrees of freedom.

```
public static double cdf (int n, double x)
```

Computes the Student t-distribution function $u = F(x)$ with n degrees of freedom. Gives 13 decimal digits of precision for $n \le 10^5$. For $n > 10^5$, gives at least 6 decimal digits of precision everywhere, and at least 9 decimal digits of precision for all $u > 10^{-15}$.

```
@Deprecated
public static double cdf2 (int n, int d, double x)
```

Same as `cdf(n, x)`.

```
public static double barF (int n, double x)
```

Computes the complementary distribution function $v = \bar{F}(x)$ with n degrees of freedom. Gives 13 decimal digits of precision for $n \le 10^5$. For $n > 10^5$, gives at least 6 decimal digits of precision everywhere, and at least 9 decimal digits of precision for all $v > 10^{-15}$.

```
public static double inverseF (int n, double u)
```

Returns the inverse $x = F^{-1}(u)$ of Student t-distribution function with n degrees of freedom. Gives 13 decimal digits of precision for $n \le 10^5$, and at least 9 decimal digits of precision for $n > 10^5$.

```
public static double[] getMLE (double[] x, int m)
```

Estimates the parameter n of the Student t-distribution using the maximum likelihood method, from the m observations $x[i]$, $i = 0, 1, \ldots, m-1$. The estimate is returned in a one-element array.

```
public static StudentDist getInstanceFromMLE (double[] x, int m)
```

Creates a new instance of a Student t-distribution with parameter n estimated using the maximum likelihood method based on the m observations $x[i]$, $i = 0, 1, \ldots, m-1$.

```
public static double getMean (int n)
```

Returns the mean $E[X] = 0$ of the Student t-distribution with parameter n.

```
public static double getVariance (int n)
```

Computes and returns the variance $\mathrm{Var}[X] = n/(n-2)$ of the Student t-distribution with parameter n.

```
public static double getStandardDeviation (int n)
```

Computes and returns the standard deviation of the Student t-distribution with parameter n.

```
public int getN()
```

Returns the parameter n associated with this object.

```
public void setN (int n)
```

Sets the parameter n associated with this object.

```
public double[] getParams ()
```

Return a table containing the parameter of the current distribution.

StudentDistQuick

Extends the class `StudentDist` for the *Student* t-distribution. Uses methods that are faster but less precise than `StudentDist`.

```
package umontreal.iro.lecuyer.probdist;

public class StudentDistQuick extends StudentDist
```

Constructors

`public StudentDistQuick (int n)`

Constructs a `StudentDistQuick` object with `n` degrees of freedom.

Methods

`public static double cdf (int n, double x)`

Returns the approximation of [29, page 96] of the Student t-distribution function with n degrees of freedom. Is very poor in the tails but good in the central part of the range.

`public static double barF (int n, double x)`

Computes the complementary distribution function $\bar{F}(x)$.

`public static double inverseF (int n, double u)`

Returns an approximation of $F^{-1}(u)$, where F is the Student t-distribution function with n degrees of freedom. Gives at least 5 decimal digits of precision when $n \geq 3$ (see [23]). Uses exact formulae for $n = 1$ and $n = 2$.

TriangularDist

Extends the class `ContinuousDistribution` for the *triangular* distribution (see [27, page 297] and [33, page 317]) with domain $[a, b]$ and *mode* (or shape parameter) m, where $a \le m \le b$. The density function is

$$f(x) = \begin{cases} \frac{2(x-a)}{(b-a)(m-a)} & \text{if } a \le x \le m, \\ \frac{2(b-x)}{(b-a)(b-m)} & \text{if } m \le x \le b, \\ 0 & \text{elsewhere,} \end{cases} \tag{104}$$

the distribution function is

$$F(x) = \begin{cases} 0 & \text{for } x < a, \\ \frac{(x-a)^2}{(b-a)(m-a)} & \text{if } a \le x \le m, \\ 1 - \frac{(b-x)^2}{(b-a)(b-m)} & \text{if } m \le x \le b, \\ 1 & \text{for } x > b, \end{cases} \tag{105}$$

and the inverse distribution function is given by

$$F^{-1}(u) = \begin{cases} a + \sqrt{(b-a)(m-a)u} & \text{if } 0 \le u \le \frac{m-a}{b-a}, \\ b - \sqrt{(b-a)(b-m)(1-u)} & \text{if } \frac{m-a}{b-a} \le u \le 1. \end{cases} \tag{106}$$

```
package umontreal.iro.lecuyer.probdist;

public class TriangularDist extends ContinuousDistribution
```

Constructors

`public TriangularDist()`

Constructs a `TriangularDist` object with default parameters $a = 0$, $b = 1$, and $m = 0.5$.

`public TriangularDist (double m)`

Constructs a `TriangularDist` object with parameters $a = 0$, $b = 1$ and $m =$ `m`.

`public TriangularDist (double a, double b, double m)`

Constructs a `TriangularDist` object with parameters a, b and m.

Methods

```
public static double density (double a, double b, double m, double x)
```

Computes the density function.

```
public static double cdf (double a, double b, double m, double x)
```

Computes the distribution function.

```
public static double barF (double a, double b, double m, double x)
```

Computes the complementary distribution function.

```
public static double inverseF (double a, double b, double m, double u)
```

Computes the inverse distribution function.

```
public static double[] getMLE (double[] x, int n, double a, double b)
```

Estimates the parameter m of the triangular distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimated parameter is returned in a one-element array: $[\hat{m}]$. See [44, 24, 31].

```
public static TriangularDist getInstanceFromMLE (double[] x, int n,
                                                 double a, double b)
```

Creates a new instance of a triangular distribution with parameters `a` and `b`. m is estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static double getMean (double a, double b, double m)
```

Computes and returns the mean $E[X] = (a + b + m)/3$ of the triangular distribution with parameters a, b, m.

```
public static double getVariance (double a, double b, double m)
```

Computes and returns the variance $\mathrm{Var}[X] = (a^2 + b^2 + m^2 - ab - am - bm)/18$ of the triangular distribution with parameters a, b, m.

```
public static double getStandardDeviation (double a, double b, double m)
```

Computes and returns the standard deviation of the triangular distribution with parameters a, b, m.

```
public double getA()
```

Returns the value of a for this object.

```
public double getB()
```

Returns the value of b for this object.

```
public double getM()
```

Returns the value of m for this object.

```
public void setParams (double a, double b, double m)
```

Sets the value of the parameters a, b and m for this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[a, b, m]$.

`public String toString ()`

Returns a `String` containing information about the current distribution.

UniformDist

Extends the class `ContinuousDistribution` for the *uniform* distribution [27, page 276] over the interval $[a, b]$. Its density is

$$f(x) = 1/(b-a) \qquad \text{for } a \le x \le b \tag{107}$$

and 0 elsewhere. The distribution function is

$$F(x) = (x-a)/(b-a) \qquad \text{for } a \le x \le b \tag{108}$$

and its inverse is

$$F^{-1}(u) = a + (b-a)u \qquad \text{for } 0 \le u \le 1. \tag{109}$$

```
package umontreal.iro.lecuyer.probdist;

public class UniformDist extends ContinuousDistribution
```

Constructors

`public UniformDist()`

Constructs a uniform distribution over the interval $(a, b) = (0, 1)$.

`public UniformDist (double a, double b)`

Constructs a uniform distribution over the interval (a, b).

Methods

`public static double density (double a, double b, double x)`

Computes the uniform density function $f(x)$ in (107).

`public static double cdf (double a, double b, double x)`

Computes the uniform distribution function as in (108).

`public static double barF (double a, double b, double x)`

Computes the uniform complementary distribution function $\bar{F}(x)$.

`public static double inverseF (double a, double b, double u)`

Computes the inverse of the uniform distribution function (109).

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

Estimates the parameter (a, b) of the uniform distribution using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[a,\ b]$.

`public static UniformDist getInstanceFromMLE (double[] x, int n)`

Creates a new instance of a uniform distribution with parameters a and b estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Computes and returns the mean $E[X] = (a + b)/2$ of the uniform distribution with parameters a and b.

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

Computes and returns the variance $\mathrm{Var}[X] = (b - a)^2/12$ of the uniform distribution with parameters a and b.

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

Computes and returns the standard deviation of the uniform distribution with parameters a and b.

`public double getA()`

Returns the parameter a.

`public double getB()`

Returns the parameter b.

`public void setParams (double a, double b)`

Sets the parameters a and b for this object.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in regular order: $[a, b]$.

WeibullDist

This class extends the class `ContinuousDistribution` for the *Weibull* distribution [26, page 628] with shape parameter $\alpha > 0$, location parameter δ, and scale parameter $\lambda > 0$. The density function is

$$f(x) = \alpha\lambda^{\alpha}(x-\delta)^{\alpha-1}e^{-(\lambda(x-\delta))^{\alpha}} \qquad \text{for } x > \delta, \tag{110}$$

the distribution function is

$$F(x) = 1 - e^{-(\lambda(x-\delta))^{\alpha}} \qquad \text{for } x > \delta, \tag{111}$$

and the inverse distribution function is

$$F^{-1}(u) = (-\ln(1-u))^{1/\alpha}/\lambda + \delta \qquad \text{for } 0 \le u < 1.$$

```
package umontreal.iro.lecuyer.probdist;

public class WeibullDist extends ContinuousDistribution
```

Constructors

```
public WeibullDist (double alpha)
```

Constructs a `WeibullDist` object with parameters α = `alpha`, $\lambda = 1$, and $\delta = 0$.

```
public WeibullDist (double alpha, double lambda, double delta)
```

Constructs a `WeibullDist` object with parameters α = `alpha`, λ = `lambda`, and δ = `delta`.

Methods

```
public static double density (double alpha, double lambda,
                              double delta, double x)
```

Computes the density function.

```
public static double density (double alpha, double x)
```

Same as `density (alpha, 1, 0, x)`.

```
public static double cdf (double alpha, double lambda,
                          double delta, double x)
```

Computes the distribution function.

```
public static double cdf (double alpha, double x)
```

Same as `cdf (alpha, 1, 0, x)`.

```
public static double barF (double alpha, double lambda,
                           double delta, double x)
```

Computes the complementary distribution function.

```
public static double barF (double alpha, double x)
```

Same as `barF (alpha, 1, 0, x)`.

```
public static double inverseF (double alpha, double lambda,
                               double delta, double u)
```

Computes the inverse of the distribution function.

```
public static double inverseF (double alpha, double x)
```

Same as `inverseF (alpha, 1, 0, x)`.

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

Estimates the parameters (α, λ) of the Weibull distribution, assuming that $\delta = 0$, using the maximum likelihood method, from the n observations $x[i]$, $i = 0, 1, \ldots, n-1$. The estimates are returned in a two-element array, in regular order: $[\alpha,\ \lambda]$.

```
public static WeibullDist getInstanceFromMLE (double[] x, int n)
```

Creates a new instance of a Weibull distribution with parameters α, λ and $\delta = 0$ estimated using the maximum likelihood method based on the n observations $x[i]$, $i = 0, 1, \ldots, n-1$.

```
public static double getMean (double alpha, double lambda, double delta)
```

Computes and returns the mean $E[X] = \delta + \Gamma(1 + 1/\alpha)/\lambda$ of the Weibull distribution with parameters α, λ and δ.

```
public static double getVariance (double alpha, double lambda,
                                  double delta)
```

Computes and returns the variance $\mathrm{Var}[X] = |\Gamma(2/\alpha + 1) - \Gamma^2(1/\alpha + 1)|/\lambda^2$ of the Weibull distribution with parameters α, λ and δ.

```
public static double getStandardDeviation (double alpha, double lambda,
                                           double delta)
```

Computes and returns the standard deviation of the Weibull distribution with parameters α, λ and δ.

```
public double getAlpha()
```

Returns the parameter α.

```
public double getLambda()
```

Returns the parameter λ.

```
public double getDelta()
```

Returns the parameter δ.

```
public void setParams (double alpha, double lambda, double delta)
```

Sets the parameters α, λ and δ for this object.

```
public double[] getParams ()
```

Return a table containing the parameters of the current distribution. This table is put in regular order: $[\alpha,\ \lambda,\ \delta]$.

TruncatedDist

This container class takes an arbitrary continuous distribution and truncates it to an interval $[a, b]$, where a and b can be finite or infinite. If the original density and distribution function are f_0 and F_0, the new ones are f and F, defined by

$$f(x) = \frac{f_0(x)}{F_0(b) - F_0(a)} \qquad \text{for } a \le x \le b$$

and $f(x) = 0$ elsewhere, and

$$F(x) = \frac{F_0(x) - F_0(a)}{F_0(b) - F_0(a)} \qquad \text{for } a \le x \le b.$$

The inverse distribution function of the truncated distribution is

$$F^{-1}(u) = F_0^{-1}(F_0(a) + (F_0(b) - F_0(a))u)$$

where F_0^{-1} is the inverse distribution function of the original distribution.

```
package umontreal.iro.lecuyer.probdist;

public class TruncatedDist extends ContinuousDistribution
```

Constructor

```
public TruncatedDist (ContinuousDistribution dist, double a, double b)
```

Constructs a new distribution by truncating distribution `dist` to the interval $[a, b]$. Restrictions: a and b must be finite.

Methods

```
public double getMean()
```

Returns an approximation of the mean computed with the Simpson 1/3 numerical integration rule.

```
public double getVariance()
```

Returns an approximation of the variance computed with the Simpson 1/3 numerical integration rule.

```
public double getStandardDeviation()
```

Returns the square root of the approximate variance.

`public double getA()`

Returns the value of a.

`public double getB()`

Returns the value of b.

`public double getFa()`

Returns the value of $F_0(a)$.

`public double getFb()`

Returns the value of $F_0(b)$.

`public double getArea()`

Returns the value of $F_0(b) - F_0(a)$, the area under the truncated density function.

`public void setParams (ContinuousDistribution dist, double a, double b)`

Sets the parameters `dist`, a and b for this object. See the constructor for details.

`public double[] getParams ()`

Return a table containing the parameters of the current distribution. This table is put in order: $[a, b, F_0(a), F_0(b), F_0(b) - F_0(a)]$.

`public String toString ()`

Returns a `String` containing information about the current distribution.

AndersonDarlingDist

Extends the class `ContinuousDistribution` for the *Anderson-Darling* distribution (see [1, 35, 38, 55]). Given a sample of n independent uniforms U_i over $(0, 1)$, the *Anderson-Darling* statistic A_n^2 is defined by

$$A_n^2 \quad = \quad -n - \frac{1}{n} \sum_{j=1}^{n} \left\{ (2j-1) \ln(U_{(j)}) + (2n+1-2j) \ln(1-U_{(j)}) \right\},$$

where the $U_{(j)}$ are the U_i sorted in increasing order. The distribution function (the cumulative probabilities) is defined as $F_n(x) = P[A_n^2 \le x]$.

```
package umontreal.iro.lecuyer.probdist;
```

```
public class AndersonDarlingDist extends ContinuousDistribution
```

Constructor

`public AndersonDarlingDist (int n)`

Constructs an *Anderson-Darling* distribution for a sample of size n.

Methods

`public static double density (int n, double x)`

Computes the density of the *Anderson-Darling* distribution with parameter n.

`public static double cdf (int n, double x)`

Computes the *Anderson-Darling* distribution function $F_n(x)$, with parameter n, using Marsaglia's and al. algorithm [38]. First the asymptotic distribution for $n \to \infty$ is computed. Then an empirical correction obtained by simulation is added for finite n.

`public static double barF (int n, double x)`

Computes the complementary distribution function $\bar{F}_n(x)$ with parameter n.

`public static double inverseF (int n, double u)`

Computes the inverse $x = F_n^{-1}(u)$ of the *Anderson-Darling* distribution with parameter n.

`public int getN()`

Returns the parameter n of this object.

`public void setN (int n)`

Sets the parameter n of this object.

`public double[] getParams ()`

Return an array containing the parameter n of the current distribution.

AndersonDarlingDistQuick

Extends the class `AndersonDarlingDist` for the *Anderson-Darling* distribution (see [1, 35, 55]). This class implements a faster version than class `AndersonDarlingDist`.

```
package umontreal.iro.lecuyer.probdist;

public class AndersonDarlingDistQuick extends AndersonDarlingDist
```

Constructor

`public AndersonDarlingDistQuick (int n)`

Constructs an *Anderson-Darling* distribution for a sample of size n.

Methods

`public static double density (int n, double x)`

Computes the density of the *Anderson-Darling* distribution with parameter n.

`public static double cdf (int n, double x)`

Computes the *Anderson-Darling* distribution function $F_n(x)$ with parameter n. The asymptotic distribution $F_\infty(x) = \lim_{n\to\infty} F_n(x)$ was first computed by numerical integration. Then a linear correction $O(1/n)$ obtained by simulation was added. The absolute error on $F_n(x)$ is estimated to be less than 0.001 for $n > 6$, except far in the tails. For $n = 2, 3, 4, 6$, it is estimated to be less than 0.04, 0.01, 0.005, 0.002, respectively. For $n = 1$, the method returns the exact value $F_1(x) = \sqrt{1 - 4e^{-x-1}}$, for $x \ge \ln(4) - 1$.

`public static double barF (int n, double x)`

Computes the complementary distribution function $\bar{F}_n(x)$ with parameter n.

`public static double inverseF (int n, double u)`

Computes the inverse $x = F_n^{-1}(u)$ of the *Anderson-Darling* distribution with parameter n.

CramerVonMisesDist

Extends the class `ContinuousDistribution` for the Cramér-von Mises distribution (see [15, 54, 55]). Given a sample of n independent uniforms U_i over $[0,1]$, the Cramér-von Mises statistic W_n^2 is defined by

$$W_n^2 = \frac{1}{12n} + \sum_{j=1}^{n} \left(U_{(j)} - \frac{(j-0.5)}{n}\right)^2, \tag{112}$$

where the $U_{(j)}$ are the U_i sorted in increasing order. The distribution function (the cumulative probabilities) is defined as $F_n(x) = P[W_n^2 \le x]$.

```
package umontreal.iro.lecuyer.probdist;

public class CramerVonMisesDist extends ContinuousDistribution
```

Constructor

`public CramerVonMisesDist (int n)`

Constructs a *Cramér-von Mises* distribution for a sample of size n.

Methods

`public static double density (int n, double x)`

Computes the density function for a *Cramér-von Mises* distribution with parameter n.

`public static double cdf (int n, double x)`

Computes the Cramér-von Mises distribution function with parameter n. Returns an approximation of $P[W_n^2 \le x]$, where W_n^2 is the Cramér von Mises statistic (see [54, 55, 1, 30]). The approximation is based on the distribution function of $W^2 = \lim_{n\to\infty} W_n^2$, which has the following series expansion derived by Anderson and Darling [1]:

$$P(W^2 \le x) \;=\; \frac{1}{\pi\sqrt{x}} \sum_{j=0}^{\infty} (-1)^j \binom{-1/2}{j} \sqrt{4j+1}\;\exp\left\{-\frac{(4j+1)^2}{16x}\right\} \mathrm{K}_{1/4}\left(\frac{(4j+1)^2}{16x}\right),$$

where K_ν is the modified Bessel function of the second kind. To correct for the deviation between $P(W_n^2 \le x)$ and $P(W^2 \le x)$, we add a correction in $1/n$, obtained empirically by simulation. For $n = 10$, 20, 40, the error is less than 0.002, 0.001, and 0.0005, respectively, while for $n \ge 100$ it is less than 0.0005. For $n \to \infty$, we estimate that the method returns at least 6 decimal digits of precision. For $n = 1$, the method uses the exact distribution: $P(W_1^2 \le x) = 2\sqrt{x - 1/12}$ for $1/12 \le x \le 1/3$.

`public static double barF (int n, double x)`

Computes the complementary distribution function $\bar{F}_n(x)$ with parameter n.

`public static double inverseF (int n, double u)`

Computes $x = F_n^{-1}(u)$, where F_n is the *Cramér-von Mises* distribution with parameter n.

`public static double getMean (int n)`

Returns the mean of the distribution with parameter n.

`public static double getVariance (int n)`

Returns the variance of the distribution with parameter n.

`public static double getStandardDeviation (int n)`

Returns the standard deviation of the distribution with parameter n.

`public int getN()`

Returns the parameter n of this object.

`public void setN (int n)`

Sets the parameter n of this object.

`public double[] getParams ()`

Return an array containing the parameter n of this object.

KolmogorovSmirnovPlusDist

Extends the class `ContinuousDistribution` for the *Kolmogorov-Smirnov+* distribution (see [12, 15, 8]). Given a sample of n independent uniforms U_i over $[0,1]$, the *Kolmogorov-Smirnov+* statistic D_n^+ and the *Kolmogorov-Smirnov−* statistic D_n^-, are defined by

$$D_n^+ = \max_{1\le j\le n} \left(j/n - U_{(j)}\right), \tag{113}$$

$$D_n^- = \max_{1\le j\le n} \left(U_{(j)} - (j-1)/n\right), \tag{114}$$

where the $U_{(j)}$ are the U_i sorted in increasing order. Both statistics follows the same distribution function, i.e. $F_n(x) = P[D_n^+ \le x] = P[D_n^- \le x]$.

```
package umontreal.iro.lecuyer.probdist;

public class KolmogorovSmirnovPlusDist extends ContinuousDistribution
```

Constructor

```
public KolmogorovSmirnovPlusDist (int n)
```

Constructs an *Kolmogorov-Smirnov+* distribution for a sample of size n.

Methods

```
public static double density (int n, double x)
```

Computes the density of the *Kolmogorov-Smirnov+* distribution with parameter n.

```
public static double cdf (int n, double x)
```

Computes the *Kolmogorov-Smirnov+* distribution function $F_n(x)$ with parameter n. The distribution function can be approximated via the following expressions:

$$P[D_n^+ \le x] = 1 - x \sum_{i=0}^{\lfloor n(1-x)\rfloor} \binom{n}{i} \left(\frac{i}{n} + x\right)^{i-1} \left(1 - \frac{i}{n} - x\right)^{n-i} \tag{115}$$

$$= x \sum_{j=0}^{\lfloor nx\rfloor} \binom{n}{j} \left(\frac{j}{n} - x\right)^{j} \left(1 - \frac{j}{n} + x\right)^{n-j-1} \tag{116}$$

$$\approx 1 - e^{-2nx^2}\left[1 - \frac{2x}{3}\left(1 - x\left(1 - \frac{2nx^2}{3}\right)\right.\right.$$
$$\left.\left. - \frac{2}{3n}\left(\frac{1}{5} - \frac{19nx^2}{15} + \frac{2n^2x^4}{3}\right)\right) + O(n^{-2})\right]. \tag{117}$$

Formula (115) and (116) can be found in [15], equations (2.1.12) and (2.1.16), while (117) can be found in [12]. Formula (116) becomes numerically unstable as nx increases. The approximation (117) is simpler to compute and excellent when nx is large. The relative error on $F_n(x) = P[D_n^+ \le x]$ is always less than 10^{-5}.

```
public static double barF (int n, double x)
```

Computes the complementary distribution function $\bar{F}_n(x)$ with parameter n.

```
public static double inverseF (int n, double u)
```

Computes the inverse $x = F^{-1}(u)$ of the distribution with parameter n.

```
public int getN()
```

Returns the parameter n of this object.

```
public void setN (int n)
```

Sets the parameter n of this object.

```
public double[] getParams ()
```

Returns an array containing the parameter n of this object.

KolmogorovSmirnovDist

Extends the class `ContinuousDistribution` for the *Kolmogorov-Smirnov* distribution with parameter n [15]. Given an empirical distribution F_n with n independent observations and a continuous distribution $F(x)$, the two-sided *Kolmogorov-Smirnov* statistic is defined as

$$D_n = \sup_{-\infty \le x \le \infty} |F_n(x) - F(x)| \;=\; \max\{D_n^+, D_n^-\}, \tag{118}$$

where D_n^+ and D_n^- are the *Kolmogorov-Smirnov*+ and *Kolmogorov-Smirnov*− statistics as defined in equations 113 and 114 on page 132 of this guide. This class implements a high precision version of the *Kolmogorov-Smirnov* distribution $P[D_n \le x]$; it is a Java translation of the C program written in [39]. According to its authors, it should give 13 decimal digits of precision. It is extremely slow for large values of n.

```
package umontreal.iro.lecuyer.probdist;

public class KolmogorovSmirnovDist extends ContinuousDistribution
```

Constructor

`public KolmogorovSmirnovDist (int n)`

Constructs a *Kolmogorov-Smirnov* distribution with parameter n. Restriction: $n \ge 1$.

Methods

`public static double density (int n, double x)`

Computes the density for the *Kolmogorov-Smirnov* distribution with parameter n.

`public static double cdf (int n, double x)`

Computes the *Kolmogorov-Smirnov* distribution function $F(x)$ with parameter n using Durbin's matrix formula [15]. It is a translation of the C program in [39]; according to its authors, it returns 13 decimal digits of precision. It is extremely slow for large n.

`public static double barF (int n, double x)`

Computes the complementary distribution function $\bar{F}(x)$ with parameter n. Simply returns `1 - cdf(n,x)`. It is not precise in the upper tail.

`public static double inverseF (int n, double u)`

Computes the inverse $x = F^{-1}(u)$ of the *Kolmogorov-Smirnov* distribution $F(x)$ with parameter n.

`public int getN()`

Returns the parameter n of this object.

`public void setN (int n)`

Sets the parameter n of this object.

`public double[] getParams ()`

Returns an array containing the parameter n of this object.

KolmogorovSmirnovDistQuick

Extends the class `KolmogorovSmirnovDist` for the *Kolmogorov-Smirnov* distribution. The methods of this class are much faster than those of class `KolmogorovSmirnovDist`.

```
package umontreal.iro.lecuyer.probdist;

public class KolmogorovSmirnovDistQuick extends KolmogorovSmirnovDist
```

Constructor

```
public KolmogorovSmirnovDistQuick (int n)
```

Constructs a *Kolmogorov-Smirnov* distribution with parameter n.

Methods

```
public static double density (int n, double x)
```

Computes the density for the *Kolmogorov-Smirnov* distribution with parameter n.

```
public static double cdf (int n, double x)
```

Computes the *Kolmogorov-Smirnov* distribution function $u = P[D_n \le x]$ with parameter n, using the program described in [53]. This method uses Pomeranz's recursion algorithm and the Durbin matrix algorithm [9, 49, 39] for $n \le 500$, which returns at least 13 decimal digits of precision. It uses the Pelz-Good asymptotic expansion [47] in the central part of the range for $n > 500$ and returns at least 7 decimal digits of precision everywhere for $500 < n \le 100000$. For $n > 100000$, it returns at least 5 decimal digits of precision for all $u > 10^{-16}$, and a few correct decimals when $u \le 10^{-16}$. This method is much faster than method `cdf` of `KolmogorovSmirnovDist` for moderate or large n. Restriction: $n \ge 1$.

```
public static double barF (int n, double x)
```

Computes the complementary *Kolmogorov-Smirnov* distribution $P[D_n \ge x]$ with parameter n, in a form that is more precise in the upper tail, using the program described in [53]. It returns at least 10 decimal digits of precision everywhere for all $n \le 500$, at least 6 decimal digits of precision for $500 < n \le 200000$, and a few correct decimal digits (1 to 5) for $n > 200000$. This method is much faster and more precise for x close to 1, than method `barF` of `KolmogorovSmirnovDist` for moderate or large n. Restriction: $n \ge 1$.

```
public static double inverseF (int n, double u)
```

Computes the inverse $x = F^{-1}(u)$ of the distribution $F(x)$ with parameter n.

WatsonGDist

Extends the class `ContinuousDistribution` for the Watson G distribution (see [13, 57]). Given a sample of n independent uniforms U_i over $[0,1]$, the G statistic is defined by

$$\begin{aligned} G_n &= \sqrt{n} \max_{1 \le j \le n} \left\{ j/n - U_{(j)} + \bar{U}_n - 1/2 \right\} \qquad (119) \\ &= \sqrt{n} \left(D_n^+ + \bar{U}_n - 1/2 \right), \end{aligned}$$

where the $U_{(j)}$ are the U_i sorted in increasing order, $\bar{U}_n$ is the average of the observations U_i, and D_n^+ is the Kolmogorov-Smirnov+ statistic. The distribution function (the cumulative probabilities) is defined as $F_n(x) = P[G_n \le x]$.

```
package umontreal.iro.lecuyer.probdist;

public class WatsonGDist extends ContinuousDistribution
```

Constructor

`public WatsonGDist (int n)`

Constructs a *Watson* distribution for a sample of size n.

Methods

`public static double density (int n, double x)`

Computes the density function for a *Watson G* distribution with parameter n.

`public static double cdf (int n, double x)`

Computes the Watson G distribution function $F_n(x)$, with parameter n. A cubic spline interpolation is used for the asymptotic distribution when $n \to \infty$, and an empirical correction of order $1/\sqrt{n}$, obtained empirically from 10^7 simulation runs with $n = 256$ is then added. The absolute error is estimated to be less than 0.01, 0.005, 0.002, 0.0008, 0.0005, 0.0005, 0.0005 for $n = 16$, 32, 64, 128, 256, 512, 1024, respectively.

`public static double barF (int n, double x)`

Computes the complementary distribution function $\bar{F}_n(x)$ with parameter n.

`public static double inverseF (int n, double u)`

Computes $x = F_n^{-1}(u)$, where F_n is the *Watson G* distribution with parameter n.

`public int getN()`

Returns the parameter n of this object.

`public void setN (int n)`

Sets the parameter n of this object.

`public double[] getParams ()`

Return an array containing the parameter n of this object.

WatsonUDist

Extends the class `ContinuousDistribution` for the *Watson U* distribution (see [15, 54, 55]). Given a sample of n independent uniforms u_i over $[0, 1]$, the *Watson* statistic U_n^2 is defined by

$$
\begin{aligned}
W_n^2 &= \frac{1}{12n} + \sum_{j=1}^{n} \left\{ u_{(j)} - \frac{(j-1/2)}{n} \right\}^2, \\
U_n^2 &= W_n^2 - n\left(\bar{u}_n - 1/2\right)^2.
\end{aligned}
$$

where the $u_{(j)}$ are the u_i sorted in increasing order, and $\bar{u}_n$ is the average of the observations u_i. The distribution function (the cumulative probabilities) is defined as $F_n(x) = P[U_n^2 \le x]$.

```
package umontreal.iro.lecuyer.probdist;

public class WatsonUDist extends ContinuousDistribution
```

Constructor

`public WatsonUDist (int n)`

Constructs a *Watson U* distribution for a sample of size n.

Methods

`public static double density (int n, double x)`

Computes the density of the *Watson U* distribution with parameter n.

`public static double cdf (int n, double x)`

Computes the Watson U distribution function, i.e. returns $P[U_n^2 \le x]$, where U_n^2 is the Watson statistic defined in (120). We use the asymptotic distribution for $n \to \infty$, plus a correction in $O(1/n)$, as given in [11].

`public static double barF (int n, double x)`

Computes the complementary distribution function $\bar{F}_n(x)$, where F_n is the *Watson U* distribution with parameter n.

`public static double inverseF (int n, double u)`

Computes $x = F_n^{-1}(u)$, where F_n is the *Watson U* distribution with parameter n.

`public static double getMean (int n)`

Returns the mean of the *Watson U* distribution with parameter n.

`public static double getVariance (int n)`

Returns the variance of the *Watson U* distribution with parameter n.

`public static double getStandardDeviation (int n)`

Returns the standard deviation of the *Watson U* distribution with parameter n.

`public int getN()`

Returns the parameter n of this object.

`public void setN (int n)`

Sets the parameter n of this object.

`public double[] getParams ()`

Return an array containing the parameter n of this object.

References

[1] T. W. Anderson and D. A. Darling. Asymptotic theory of certain goodness of fit criteria based on stochastic processes. *Annals of Mathematical Statistics*, 23:193–212, 1952.

[2] D. J. Best and D. E. Roberts. Algorithm AS 91: The percentage points of the χ^2 distribution. *Applied Statistics*, 24:385–388, 1975.

[3] G. P. Bhattacharjee. The incomplete gamma integral. *Applied Statistics*, 19:285–287, 1970. AS32.

[4] Z. W. Birnbaum and S. C. Saunders. A new family of life distributions. *Journal of Applied Probability*, 6:319–327, 1969.

[5] J. M. Blair, C. A. Edwards, and J. H. Johnson. Rational Chebyshev approximations for the inverse of the error function. *Mathematics of Computation*, 30:827–830, 1976.

[6] L. N. Bol'shev. Some applications of Pearson transformations. *Review of the Internat. Stat. Institute*, 32:14–16, 1964.

[7] P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation*. Springer-Verlag, New York, NY, second edition, 1987.

[8] J. R. Brown and M. E. Harvey. Rational arithmetic Mathematica functions to evaluate the one-sided one-sample K-S cumulative sample distribution. *Journal of Statistical Software*, 19(6):1–32, 2007.

[9] J. R. Brown and M. E. Harvey. Rational arithmetic Mathematica functions to evaluate the two-sided one sample K-S cumulative sample distribution. *Journal of Statistical Software*, 26(2):1–40, 2008.

[10] B. H. Camp. Approximation to the point binomial. *Ann. Math. Stat.*, 22:130–131, 1951.

[11] S. Csörgő and J. J. Faraway. The exact and asymptotic distributions of Cramér-von mises statistics. *Journal of the Royal Statistical Society, Series B*, 58:221–234, 1996.

[12] D. A. Darling. On the theorems of Kolmogorov-Smirnov. *Theory of Probability and Its Applications*, V(4):356–360, 1960.

[13] D. A. Darling. On the asymptotic distribution of Watson's statistic. *The Annals of Statistics*, 11(4):1263–1266, 1983.

[14] G. Derflinger, W. Hörmann, and J. Leydold. Random variate generation by numerical inversion when only the density is known. *Preprint of the Department of Statistics and Mathematics 78, Wirtschaftsuniversität Wien, Austria*, 2008. See `http://epub.wu-wien.ac.at/english/`.

[15] J. Durbin. *Distribution Theory for Tests Based on the Sample Distribution Function*. SIAM CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, Philadelphia, PA, 1973.

[16] M. Evans, N. Hastings, and B. Peacock. *Statistical Distributions.* Wiley, 3rd edition, 2000.

[17] M. Evans and T. Swartz. *Approximating Integrals via Monte Carlo and Deterministic Methods.* Oxford University Press, Oxford, UK, 2000.

[18] W. Gautschi. Algorithm 222: Incomplete beta function ratios. *Communications of the ACM*, 7(3):143–144, 1964.

[19] W. Gautschi. Certification of algorithm 222: Incomplete beta function ratios. *Communications of the ACM*, 7(3):244, 1964.

[20] J. E. Gentle. *Random Number Generation and Monte Carlo Methods.* Springer, New York, NY, 1998.

[21] I. B. Gertsbakh and Y. Shpungin. *Models of Network Reliability.* CRC Press, Boca Raton, FL, 2010.

[22] R. B. Goldstein. Algorithm 451: Chi-square quantiles. *Communications of the ACM*, 16:483–485, 1973.

[23] G. W. Hill. Algorithm 395: Student's t-distribution. *Communications of the ACM*, 13:617–619, 1970.

[24] J. S. Huang and P. S. Shen. More maximum likelihood oddities. *Journal of Statistical Planning and Inference*, 137(7):2151–2155, 2007.

[25] N. L. Johnson and S. Kotz. *Distributions in Statistics: Discrete Distributions.* Houghton Mifflin, Boston, 1969.

[26] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume 1. Wiley, 2nd edition, 1994.

[27] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume 2. Wiley, 2nd edition, 1995.

[28] V. Kachitvichyanukul and B. Schmeiser. Computer generation of hypergeometric random variates. *J. Statist. Comput. Simul.*, 22:127–145, 1985.

[29] W. J. Kennedy Jr. and J. E. Gentle. *Statistical Computing.* Dekker, New York, NY, 1980.

[30] M. Knott. The distribution of the Cramér-von Mises statistic for small sample sizes. *Journal of the Royal Statistical Society B*, 36:430–438, 1974.

[31] S. Kotz and J. R. van Dorp. *BEYOND BETA, Other Continuous Families of Distributions with Bounded Support and Applications.* World Scientific Publishing co., Singapore, 2004.

[32] G. Latouche and V. Ramaswami. *Introduction to matrix analytic methods in stochastic modeling*. Society for Industrial and Applied Mathematics, Philadelphia, Pa., 1999.

[33] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, NY, third edition, 2000.

[34] F. C. Leone, L. S. Nelson, and R. B. Nottingham. The folded normal distribution. *Technometrics*, 3(4):543–550, 1961.

[35] P. A. W. Lewis. Distribution of the Anderson-Darling statistic. *Annals of Mathematical Statistics*, 32:1118–1124, 1961.

[36] J. Leydold and W. Hörmann. *UNU.RAN—A Library for Universal Non-Uniform Random Number Generators*, 2002. Available at `http://statistik.wu-wien.ac.at/unuran`.

[37] K. V. Mardia and P. J. Zemroch. *Tables of the F and Related Distributions with Algorithms*. Academic Press, London, 1978.

[38] G. Marsaglia and J. Marsaglia. Evaluating the Anderson-Darling distribution. *Journal of Statistical Software*, 9(2):1–5, 2004. See `http://www.jstatsoft.org/v09/i02/`.

[39] G. Marsaglia, W. W. Tsang, and J. Wang. Evaluating Kolmogorov's distribution. *Journal of Statistical Software*, 8(18):1–4, 2003. URL is `http://www.jstatsoft.org/v08/i18/`.

[40] G. Marsaglia, A. Zaman, and J. C. W. Marsaglia. Rapid evaluation of the inverse normal distribution function. *Statistics and Probability Letters*, 19:259–266, 1994.

[41] W. Molenaar. *Approximations to the Poisson, Binomial and Hypergeometric Distribution Functions*, volume 31 of *Mathematical Center Tract*. Mathematisch Centrum, Amsterdam, 1970.

[42] S. L. Moshier. Cephes math library, 2000. See `http://www.moshier.net`.

[43] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models*. John Hopkins, University Press, Baltimore, 1981.

[44] E. H. Oliver. A maximum likelihood oddity. *The American Statistician*, 26(3):43–44, 1972.

[45] E. S. Pearson. Note on an approximation to the distribution of non-central χ^2. *Biometrika*, 46:364, 1959.

[46] D. B. Peizer and J. W. Pratt. A normal approximation for binomial, F, beta, and other common related tail probabilities. *Journal of the American Statistical Association*, 63:1416–1456, 1968.

[47] W. Pelz and I. J. Good. Approximating the lower tail-areas of the Kolmogorov-Smirnov one-sample statistic. *Journal of the Royal Statistical Society B*, 38(2):152–156, 1976.

[48] S. Penev and T. Raykov. A Wiener germ approximation of the noncentral chi square distribution and of its quantiles. *Computational Statistics*, 15(2):219–228, 2000.

[49] J. Pomeranz. Exact cumulative distribution of the Kolmogorov-Smirnov statistic for small samples (algorithm 487). *Communications of the ACM*, 17(12):703–704, 1974.

[50] S. M. Ross. *Introduction to Probability Models*. Academic Press, ninth edition, 2007.

[51] R. B. Schnabel. *UNCMIN—Unconstrained Optimization Package, FORTRAN*. University of Colorado at Boulder. See `http://www.ici.ro/camo/unconstr/uncmin.htm`.

[52] J. L. Schonfelder. Chebyshev expansions for the error and related functions. *Mathematics of Computation*, 32:1232–1240, 1978.

[53] R. Simard and P. L'Ecuyer. Computing the two-sided Kolmogorov-Smirnov distribution. *Journal of Statistical Software*, 39(11), 2011. URL is `http://www.jstatsoft.org/v39/i11`.

[54] M. A. Stephens. Use of the Kolmogorov-Smirnov, Cramér-Von Mises and related statistics without extensive tables. *Journal of the Royal Statistical Society, Series B*, 33(1):115–122, 1970.

[55] M. S. Stephens. Tests based on EDF statistics. In R. B. D'Agostino and M. S. Stephens, editors, *Goodness-of-Fit Techniques*. Marcel Dekker, New York and Basel, 1986.

[56] S. P. Verrill. *UNCMIN—Unconstrained Optimization Package, Java*. US Forest Service, Forest Products Laboratory. Available at `http://www1.fpl.fs.fed.us/optimization.html`.

[57] G. S. Watson. Optimal invariant tests for uniformity. In *Studies in Probability and Statistics*, pages 121–127. North Holland, Amsterdam, 1976.