SSJ User's Guide

Package `probdist`

Probability Distributions

Version: April 25, 2007

This package provides tools to compute densities, mass functions, distribution functions and their inverses, and reliability functions, for various continuous and discrete probability distributions.

Contents

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 does not generate random numbers; for that, see the package `randvar`.

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 using the interface `Distribution` (and sometimes `ContinuousDistribution`) are required by some methods in the classes `GofStat` and `GofFormat`, in package `gof`.

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);
```

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

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

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

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

Computes and 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.

DiscreteDistribution

Classes implementing discrete distributions over a *finite set of real numbers* should inherit from this class. For discrete distributions over integers, see `DiscreteDistributionInt`.

We assume that the random variable X of interest can take one of the n values $x_0 < \cdots < x_{n-1}$ (which are *sorted* by increasing order). It takes 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.

Note that the default implementation of the complementary distribution function returns `1.0 - cdf(x - 1)`, which is not accurate when $F(x)$ is near 1.

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


public class DiscreteDistribution implements Distribution
```

Constructor

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

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

Methods

`public double prob (int k)`

Returns p_k, the probability of the k-th observation, for $0 \leq k < n$. The result should be a real number in the interval $[0, 1]$.

`public double getMean()`

Computes the mean $E[X] = \sum_{i=1}^{n} p_i x_i$ of the distribution.

`public double getVariance()`

Computes the variance $\mathrm{Var}[X] = \sum_{i=1}^{n} p_i (x_i - E[X])^2$ of the distribution.

`public double getStandardDeviation()`

Computes the standard deviation of the distribution.

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.

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, which should be a real number in the interval $[0, 1]$.

```
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.

The default implementation returns `1.0 - cdf(x - 1)`, which is not accurate when $F(x)$ is near 1.

`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 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 of $\bar{F}(x)$ and of $F^{-1}(u)$, the latter using binary search to find the inverse of a generic distribution function F. The integer `decPrec` defines the target number of decimals of precision when approximating a distribution function, but there is *no guarantee* that this target is always attained.

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

public abstract class ContinuousDistribution implements Distribution

   public int decPrec = 15;

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

Returns $f(x)$, the density of X evaluated at 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 $F(a) < u < F(b)$. The calculations are done with an approximate precision of `tol`. Returns $x = F^{-1}(u)$.

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 [20].

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).

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


public class DistributionFactory
```

```
public static ContinuousDistribution getDistribution (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 getDistribution (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 ContinuousDistribution getDistribution (Class 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 DiscreteDistributionInt getDistribution (Class 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. $\boxed{1}$

`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.

[1] From Pierre: So the user must always verify if `null` was returned?

BinomialDist

Extends the class `DiscreteDistributionInt` for the *binomial* distribution [19, 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

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x} = \frac{n!}{x!(n-x)!}\, p^x (1-p)^{n-x} \qquad \text{for } x = 0, 1, 2, \ldots n, \tag{6}$$

and its distribution function is

$$F(x) = \sum_{j=0}^{x} \binom{n}{j}\, p^j (1-p)^{n-j} \qquad \text{for } x = 0, 1, 2, \ldots n, \tag{7}$$

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

public class BinomialDist extends DiscreteDistributionInt
```

Constant

```
public static double MAXN = 100000;
```

The value of the parameter n above which the tables are *not* precomputed by the constructor.

Constructor

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

Creates an object that contains the binomial terms (6), 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 mass function $p(x)$.

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

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

$$f(x) = \binom{n}{x} p^x q^{n-x} = \frac{n!}{x!(n-x)!}\, p^x q^{n-x}, \tag{8}$$

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.

If $n \le 10000$, the non-negligible terms of the sum are added explicitly. If $n > 10000$ and $np(1-p) > 100$, the Camp-Paulson normal approximation [7, 23] is used, otherwise, a Poisson approximation due to Bol'shev [5, 23] is used.

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

Computes the inverse of the binomial distribution, $x = F^{-1}(u)$, using a linear search starting at the mode if n is small. If n is larger than 10000, the linear search starts from 0 and the `cdf` static method is used to compute $F(x)$ at different values of x, which much is less efficient.

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

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

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

Estimates the parameters $[\hat{n}, \hat{p}]$ of the binomial distribution using the maximum likelihood method based on the m observations in table $x[i]$, $i = 0, 1, \ldots, m-1$.

The equations of maximum likelihood are defined in [14, page 57]:

$$\begin{aligned} \hat{n}\hat{p} &= \bar{X} \\ \sum_{i=0}^{R-1} \frac{f_i}{\hat{n} - i} &= -m \ln\left(1 - \frac{\bar{X}}{\hat{n}}\right) \end{aligned}$$

where f_i = number of x's which exceed i and $R = \max\,(x_1, \ldots, x_m)$.

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

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

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

Estimates the parameter $[\hat{p}]$ of the binomial distribution with given parameter n using 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 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 [19, page 322] with parameter p, where $0 < p < 1$. Its mass function is

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

The distribution function is given by

$$F(x) = 1 - (1-p)^{x+1}, \qquad \text{for } x = 0, 1, 2, \ldots \tag{10}$$

and its inverse is

$$F^{-1}(u) = \left\lfloor \frac{\ln(1-u)}{\ln(1-p)} \right\rfloor, \qquad \text{for } 0 \le u < 1. \tag{11}$$

```
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 probability mass function $p(x)$ given by (9) .

```
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 $\bar{F}(x)$.

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

Computes the inverse of the geometric distribution, given by (11).

```
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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameter [$\hat{p}$] of the geometric distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equation of the maximum likelihood is defined in [19, page 323]:

$$\hat{p} = \frac{1}{\bar{X} + 1}$$

`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.

HypergeometricDist

Extends the class `DiscreteDistributionInt` for the *hypergeometric* distribution [11, 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

$$p(x) = \frac{\binom{m}{x}\binom{l-m}{k-x}}{\binom{l}{k}} \qquad \text{for } \max(0, k - l + m) \le x \le \min(k, m). \tag{12}$$

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

```
public class HypergeometricDist extends DiscreteDistributionInt
```

Constant

`public static double MAXN = 100000;`

If the number of integers in the interval $[\max(0, k - l + m), \min(k, m)]$ is larger than this constant, the tables will *not* be precomputed by the constructor.

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 probability mass function $p(x)$ given by (12).

`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 $\bar{F}(x)$.

`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 [17].

`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 $\mathrm{Var}[X] = \frac{\frac{km}{l}(1-\frac{m}{l})(l-k)}{l-1}$ 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 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

$$p(x) = \frac{-1}{\log(1-\theta)} \frac{\theta^x}{x} \qquad \text{for } x = 1, 2, 3, \ldots \tag{13}$$

Its distribution function is

$$F(x) = \frac{-1}{\log(1-\theta)} \sum_{i=1}^{x} \frac{\theta^i}{i}, \qquad \text{for } x = 1, 2, 3, \ldots \tag{14}$$

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 probability mass function $p(x)$.

`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 $\bar{F}(x)$.

`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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameter $[\hat{\theta}]$ of the logarithmic distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equation of the maximum likelihood is defined in [8, page 122]:

$$\bar{X} = \frac{-\hat{\theta}}{(1-\hat{\theta})\ln(1-\hat{\theta})}$$

`public static double getMean (double theta)`

Computes and returns the mean $E[X] = \frac{-1}{\ln(1-\theta)}\frac{\theta}{(1-\theta)}$ 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.

NegativeBinomialDist

Extends the class `DiscreteDistributionInt` for the *negative binomial* distribution [19, page 324] with real parameters γ and p, where $\gamma > 0$ and $0 \le p \le 1$. Its mass function is

$$p(x) = \frac{\Gamma(\gamma + x)}{x!\,\Gamma(\gamma)} p^{\gamma}(1-p)^{x}, \qquad \text{for } x = 0, 1, 2, \ldots \tag{15}$$

where Γ is the gamma function.

If γ is an integer, $p(x)$ can be interpreted as the probability of having x failures before the γ-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
```

Constant

```
public static double MAXN = 100000;
```

If the maximum term is greater than this constant, then the tables will *not* be precomputed.

Constructor

```
public NegativeBinomialDist (double gamma, double p)
```

Creates an object that contains the probability terms (15) and the distribution function for the negative binomial distribution with parameters γ and p.

Methods

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

Computes the probability mass function defined in (15).

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

Computes the distribution function.

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

Computes the inverse function without precomputing tables. This method computes the CDF at the mode (maximum term) and performs a linear search from that point.

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

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

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

Estimates and returns the parameter $[\hat{p}]$ of the negative binomial distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$. The parameter $\gamma =$ `gamma` is assumed known. The equation of maximum likelihood is defined as $\hat{p} = \gamma/(\gamma + \bar{X})$.

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

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

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

Estimates and returns the parameters $[\hat{\gamma}, \hat{p}]$ of the negative binomial distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [14, page 132]:

$$\begin{aligned} \frac{\hat{\gamma}(1-\hat{p})}{\hat{p}} &= \bar{X} \\ \sum_{j=1}^{\infty} \frac{F_j}{(\hat{\gamma} + j - 1)} &= -n\ln(\hat{p}) \end{aligned}$$

where $F_j = \sum_{i=j}^{\infty} f_i =$ number of $x_i \geq j$.

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

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

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

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

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

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

```
public double getGamma()
```

Returns the parameter γ of this object.

`public double getP()`

Returns the parameter p of this object.

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

Sets the parameter γ and p of this object.

PascalDist

The *Pascal* distribution is a special case of the *negative binomial* distribution [19, page 324] with parameters n and p, where n is a positive integer and $0 \le p \le 1$. Its mass function is

$$p(x) = \binom{n+x-1}{x} p^n (1-p)^x, \qquad \text{for } x = 0, 1, 2, \ldots \tag{16}$$

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 (16) and the distribution function for the Pascal distribution with parameter n and p.

Methods

```
public static NegativeBinomialDist 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 in table $x[i]$, $i = 0, 1, \ldots, m-1$.

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

Estimates and returns the parameters [$\hat{n}$, $\hat{p}$] of the Pascal distribution using the maximum likelihood method based on the m observations in table $x[i]$, $i = 0, 1, \ldots, m-1$.

The equations of the maximum likelihood are defined in [14, page 132]:

$$\begin{aligned} \frac{\hat{n}(1-\hat{p})}{\hat{p}} &= \bar{X} \\ \ln(1+\hat{p}) &= \sum_{j=1}^{\infty} \frac{F_j}{(\hat{n}+j-1)} \end{aligned}$$

where $F_j = \sum_{i=j}^{\infty} f_i$ = proportion of x's which are greater than or equal to j.

```
public int getN()
```

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 [19, page 325] with mean $\lambda \geq 0$. The mass function is

$$p(x) = \frac{e^{-\lambda}\lambda^x}{x!}, \qquad \text{for } x = 0, 1, \dots \tag{17}$$

and the distribution function is

$$F(x) = e^{-\lambda} \sum_{j=0}^{x} \frac{\lambda^j}{j!}, \qquad \text{for } x = 0, 1, \dots. \tag{18}$$

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
```

Constant

```
public static double MAXLAMBDA = 100000;
```

The value of the parameter λ above which the tables are *not* precomputed by the constructor.

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 value of the Poisson probability $p(x)$, for $\lambda =$ `lambda`. If $\lambda \geq 20$, this (static) method uses the logarithm of the gamma function, defined in (40), to estimate the density.

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

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

To compute $F(x)$, all non-negligible terms of the sum are added if $\lambda \leq 150$; otherwise, the relationship $F_\lambda(x) = 1 - G_{x+1}(\lambda)$ is used, where G_{x+1} is the gamma distribution function with parameter $\alpha = x + 1$ (see `GammaDist`).

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

Computes and returns the value of the complementary Poisson distribution function, $\bar{F}(x)$, for $\lambda =$ `lambda`. Computes and adds the non-negligible terms in the tail.

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

Performs a linear search to get the inverse function without precomputed tables. If $\lambda > 150$, uses the gamma distribution by calling `GammaDist.inverseF (x+1, 15, lambda)` (see the static `cdf` method).

`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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameter $[\hat{\lambda}]$ of the Poisson distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equation of the maximum likelihood is defined by $\hat{\lambda} = \bar{X}$ (see [19, page 326]).

`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 $\mathrm{Var}[X] = \lambda$ 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.

UniformIntDist

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

$$p(x) = \frac{1}{j - i + 1} \qquad \text{for } x = i, i+1, \ldots, j \tag{19}$$

and 0 elsewhere. The distribution function is

$$F(x) = \begin{cases} 0, & \text{for } x < i \\ \dfrac{\lfloor x \rfloor - i + 1}{j - i + 1}, & \text{for } i \le x < j \\ 1, & \text{for } x \ge j. \end{cases} \tag{20}$$

and its inverse is

$$F^{-1}(u) = i + \lfloor (j - i + 1)u \rfloor \qquad \text{for } 0 \le u \le 1. \tag{21}$$

```
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 density function $f(x)$ in (19).

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

Computes the discrete uniform distribution function as in (20).

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

Computes the discrete uniform complementary distribution function $\bar{F}(x)$.

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

Computes the inverse of the discrete uniform distribution function (21).

`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 in table $x[k]$, $k = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters [î, ĵ] of the uniform distribution over integers using the maximum likelihood method based on the n observations in table $x[k]$, $k = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [19, page 320]:

$$\begin{aligned} \hat{\imath} &= \min\{X_i\} \\ \hat{\jmath} &= \max\{X_i\} \end{aligned}$$

`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.

EmpiricalDist

Extends `DiscreteDistribution` to an *empirical* distribution function, based on the observations $X_{(1)}, \ldots, X_{(n)}$ (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. [2] 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 $n/2^{\text{th}}$ 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 $n/2^{\text{th}}$ 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 $X_{(i)}$.

`public double getSampleMean()`

Returns the sample mean of the observations.

[2] The method `java.util.Arrays.sort` may be used to sort 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.

BetaDist

Extends the class `ContinuousDistribution` for the *beta* distribution [16, page 210] with shape parameters $\alpha > 0$ and $\beta > 0$, over the interval (a, b), where $a < b$. This distribution has density

$$f(x) = \frac{(x-a)^{\alpha-1}(b-x)^{\beta-1}}{\mathcal{B}(\alpha,\beta)(b-a)^{\alpha+\beta-1}}, \qquad \text{for } a < x < b, \text{ and 0 elsewhere,} \tag{22}$$

and distribution function

$$F(x) = I_{\alpha,\beta}(x) = \int_a^x \frac{(\xi-a)^{\alpha-1}(b-\xi)^{\beta-1}}{\mathcal{B}(\alpha,\beta)(b-a)^{\alpha+\beta-1}} d\xi, \qquad \text{for } a < x < b, \tag{23}$$

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

$$\mathcal{B}(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}, \tag{24}$$

and $\Gamma(x)$ is the gamma function defined in (40).

```
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.

If $\max(\alpha, \beta) \leq 1000$, uses a recurrence relation in α and β given in [9, 10]. Otherwise, if $\min(\alpha, \beta) \leq 30$, uses an approximation due to Bol'shev [21], else uses a normal approximation [25]. The method tries its best to return `d` decimals digits of precision. For `d` ≤ 13, when the normal approximation is *not* used, `d` gives a good idea of the precision attained.

```
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 [24]. 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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{\alpha},\ \hat{\beta}]$ of the beta distribution over the interval $[0, 1]$ using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [16, page 223]:

$$\begin{aligned}\psi(\alpha) - \psi(\alpha+\beta) &= \frac{1}{n}\sum_{i=1}^{n} \ln(X_i)\\ \psi(\beta) - \psi(\alpha+\beta) &= \frac{1}{n}\sum_{i=1}^{n} \ln(1-X_i)\end{aligned}$$

where ψ is the logarithmic derivative of the Gamma function $\psi(x) = \Gamma'(x)/\Gamma(x)$.

`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.

`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.

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

Computes the standard deviation of the beta distribution with parameters α and β.

`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.

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)`

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

`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 [25].

`public static BetaDist 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameter $[\hat{\alpha}]$ of the symmetrical beta distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equation of the maximum likelihood is

$$\psi(\alpha) - \psi(2\alpha) = \frac{1}{2n} \sum_{i=1}^{n} \ln(X_i(1 - X_i))$$

where ψ is the logarithmic derivative of the Gamma function $\psi(x) = \Gamma'(x)/\Gamma(x)$.

`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, $\text{Var}[X] = 1/(8\alpha + 4)$, 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.

CauchyDist

Extends the class `ContinuousDistribution` for the *Cauchy* distribution [15, page 299] with location parameter α and scale parameter $\beta > 0$. The density function is given by

$$f(x) = \frac{\beta}{\pi[(x-\alpha)^2 + \beta^2]}, \qquad \text{for } -\infty < x < \infty. \tag{25}$$

The distribution function is

$$F(x) = \frac{1}{2} + \frac{\arctan((x-\alpha)/\beta)}{\pi}, \qquad \text{for } -\infty < x < \infty, \tag{26}$$

and its inverse is

$$F^{-1}(u) = \alpha + \beta\tan(\pi(u - 1/2)). \qquad \text{for } 0 < u < 1. \tag{27}$$

```
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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{\alpha}, \hat{\beta}]$ of the Cauchy distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The estimate of the parameters is given by maximizing numerically the log-likelihood function, using the Uncmin package [26, 28].

`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.

ChiDist

Extends the class `ContinuousDistribution` for the *chi* distribution [15, 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

$$f(x) = \frac{e^{-x^2/2} x^{\nu-1}}{2^{(\nu/2)-1}\Gamma(\nu/2)}, \qquad \text{for } x > 0, \tag{28}$$

where $\Gamma(x)$ is the gamma function defined in (40). The distribution function is

$$F(x) = \frac{1}{\Gamma(\nu/2)} \int_0^{x^2/2} t^{\nu/2-1} e^{-t}\, dt. \tag{29}$$

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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

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

`public static double getMean (int nu)`

Computes and returns the mean $E[X] = \sqrt{2}\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})}$ of the chi distribution with parameter ν.

`public static double getVariance (int nu)`

Computes and returns the variance $\text{Var}[X] = 2\frac{\Gamma(\frac{\nu}{2})\Gamma(1+\frac{\nu}{2})-\Gamma^2(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})}$ 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.

ChiSquareDist

Extends the class `ContinuousDistribution` for the *chi-square* distribution with n degrees of freedom, where n is a positive integer [15, page 416]. Its density is

$$f(x) = \frac{x^{(n/2)-1}e^{-x/2}}{2^{n/2}\Gamma(n/2)}, \qquad \text{for } x > 0 \tag{30}$$

where $\Gamma(x)$ is the gamma function defined in (40). 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 (30) for a *chi-square* distribution with n degrees of freedom.

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

Computes an approximation of the chi-square distribution function with n degrees of freedom. Uses the approximation given in [18, page 116] for $n \le 350$ (which gives nearly 12 decimal digits of precision for $10^{-5} < u < 1 - 10^{-5}$). For $n > 350$, this implementation invokes the method `GammaDist.cdf (n/2, d, x/2)` instead, because it is faster and as accurate as the above approximation for such n.

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

Computes the complementary chi-square distribution function with n degrees of freedom. Uses the approximation given in [18, page 116] for $n \le 350$, and invokes the method `GammaDist.barF (n/2, d, x/2)` for $n > 350$.

```
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 [1] and in Figure L.23 of [6]. 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 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 in table $x[i]$, $i = 0, 1, \ldots, m - 1$.

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

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

`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)`

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

`public static double getStandardDeviation (int n)`

Computes and 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.

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 [6] over most of the range. For $u < 0.02$ or $u > 0.98$, it uses the approximation given in [12] for $n \ge 10$, 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 $n \ge 10$ 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 $d = 10$.

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 GammaDist 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{k}, \hat{\lambda}]$ of the Erlang distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are the same as the equations of the gamma distribution. The k parameter is the rounded value of the α parameter of the gamma distribution, and the λ parameter is equal to the β parameter of the gamma distribution.

`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, $\text{Var}[X] = k/\lambda^2$, 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.

ExponentialDist

Extends the class `ContinuousDistribution` for the *exponential* distribution [15, page 494] with mean $1/\lambda$ where $\lambda > 0$. Its density is

$$f(x) = \lambda e^{-\lambda x} \qquad \text{for } x \geq 0, \tag{31}$$

its distribution function is

$$F(x) = 1 - e^{-\lambda x}, \qquad \text{for } x \geq 0, \tag{32}$$

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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameter $[\hat{\lambda}]$ of the exponential distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equation of the maximum likelihood is defined as $\hat{\lambda} = \bar{X}^{-1}$ (see [15, page 506]).

`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.

ExtremeValueDist

Extends the class `ContinuousDistribution` for the *extreme value* (or *Gumbel*) distribution [16, page 2], with location parameter α and scale parameter $\lambda > 0$. It has density

$$f(x) = \lambda e^{-\lambda(x-\alpha)} e^{-e^{-\lambda(x-\alpha)}}, \qquad \text{for } -\infty < x < \infty, \tag{33}$$

distribution function

$$F(x) = e^{-e^{-\lambda(x-\alpha)}} \qquad \text{for } -\infty < x < \infty, \tag{34}$$

and inverse distribution function

$$F^{-1}(u) = -\ln(-\ln(u))/\lambda + \alpha, \qquad \text{for } 0 \le u \le 1. \tag{35}$$

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

public class ExtremeValueDist extends ContinuousDistribution
```

Constructors

`public ExtremeValueDist()`

Constructs a `ExtremeValueDist` object with parameters $\alpha = 0$ and $\lambda = 1$.

`public ExtremeValueDist (double alpha, double lambda)`

Constructs a `ExtremeValueDist` object with parameters $\alpha =$ `alpha` and $\lambda =$ `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)`

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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{\alpha},\ \hat{\lambda}]$ of the extreme value distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [8, page 89]:

$$\begin{aligned}
\hat{\lambda} &= \bar{X} - \frac{\sum_{i=1}^{n} X_i\, e^{-\hat{\lambda} X_i}}{\sum_{i=1}^{n} e^{-\hat{\lambda} X_i}} \\
\hat{\alpha} &= -\frac{1}{\hat{\lambda}} \ln\left(\frac{1}{n}\sum_{i=1}^{n} e^{-\hat{\lambda} X_i}\right)
\end{aligned}$$

`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.

FatigueLifeDist

Extends the class `ContinuousDistribution` for the *Fatigue Life* distribution [3] with location parameter μ, scale parameter β and shape parameter γ. Its density is

$$f(x) = \left[\frac{\sqrt{\frac{x-\mu}{\beta}} + \sqrt{\frac{\beta}{x-\mu}}}{2\gamma(x-\mu)}\right] \phi\left(\frac{\sqrt{\frac{x-\mu}{\beta}} - \sqrt{\frac{\beta}{x-\mu}}}{\gamma}\right), \qquad \text{for } x > \mu, \tag{36}$$

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

$$F(x) = \Phi\left(\frac{\sqrt{\frac{x-\mu}{\beta}} - \sqrt{\frac{\beta}{x-\mu}}}{\gamma}\right), \qquad \text{for } x > \mu, \tag{37}$$

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 (36) 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[] getMaximumLikelihoodEstimate (double[] x, int n,
                                                     double mu)
```

Estimates and returns the parameters $[\hat{\beta}, \hat{\gamma}]$ of the fatigue life distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The estimate of the parameters is given by maximizing numerically the log-likelihood function, using the Uncmin package [26, 28].

```
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.

FisherFDist

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

$$f(x) = \frac{\Gamma(\frac{n+m}{2}) n^{\frac{n}{2}} m^{\frac{m}{2}}}{\Gamma(\frac{n}{2})\Gamma(\frac{m}{2})} \frac{x^{\frac{n-2}{2}}}{(m+nx)^{\frac{n+m}{2}}}, \qquad \text{for } x > 0 \tag{38}$$

where $\Gamma(x)$ is the gamma function defined in (40).

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 FisherFDist extends ContinuousDistribution
```

Constructor

`public FisherFDist (int n, int m)`

Constructs a Fisher F-distribution with n and m degrees of freedom.

Methods

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

Computes the density function (38) for a Fisher F-distribution with n and m degrees of freedom.

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

Computes the distribution function of the Fisher F-distribution with parameters n and m, evaluated at x, with roughly d decimal digits of precision.

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

Computes the complementary distribution function of the Fisher F-distribution with parameters n and m, evaluated at x, with roughly d decimal digits of precision.

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

Computes the inverse of the Fisher F-distribution with parameters n and m, evaluated at x, with roughly d decimal digits of precision.

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

Computes and returns the mean $E[X] = m/(m-2)$ of the Fisher F-distribution with parameters n and m.

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

Computes and returns the variance $\mathrm{Var}[X] = \frac{2m^2(m+n-2)}{n(m-2)^2(m-4)}$ of the Fisher F-distribution with parameters n and m.

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

Computes and returns the standard deviation of the Fisher F-distribution with parameters n and m.

`public int getN()`

Returns the parameter n of this object.

`public int getM()`

Returns the parameter m of this object.

`public void setParams (int n, int m)`

Sets the parameters n and m of this object.

GammaDist

Extends the class `ContinuousDistribution` for the *gamma* distribution [15, page 337] with shape parameter $\alpha > 0$ and scale parameter $\lambda > 0$. The density is

$$f(x) = \frac{\lambda^\alpha x^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)}, \qquad \text{for } x > 0, \tag{39}$$

where Γ is the gamma function, defined by

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx. \tag{40}$$

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.

`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 (39). The approximation is an improved version of the algorithm in [2]. 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 using the algorithm implemented in [24] . Starting with the approximation $x = \alpha t^3$, where $t = 1 - z - \Phi^{-1}(u)\sqrt{z}$, $z = \alpha/9$, and Φ^{-1} is the inverse of the standard normal distribution, the method uses Newton iterations to estimate the inverse. The precision of the algorithm depends on the accuracy of the `barF` function. The argument `d` gives a good idea of the precision attained.

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

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

`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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters [$\hat\alpha$, $\hat\lambda$] of the gamma distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [15, page 361]:

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n}\ln(X_i) - \ln(\bar X) &= \psi(\hat\alpha) - \ln(\hat\alpha)\\ \hat\lambda\bar X &= \hat\alpha\end{aligned}$$

where ψ is the digamma function.

`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)`

HyperbolicSecantDist

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

$$f(x) = \frac{1}{2\sigma}\,\mathrm{sech}\left(\frac{\pi}{2}\frac{(x-\mu)}{\sigma}\right) \tag{41}$$

The distribution function is given by

$$F(x) = \frac{2}{\pi}\tan^{-1}\left[\exp\left(\frac{\pi}{2}\frac{(x-\mu)}{\sigma}\right)\right] \tag{42}$$

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 (41) 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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{\mu}, \hat{\sigma}]$ of the hyperbolic secant distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The estimate of the parameters is given by maximizing numerically the log-likelihood function, using the Uncmin package [26, 28].

`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.

InverseGaussianDist

Extends the class `ContinuousDistribution` for the *inverse Gaussian* distribution with location parameter $\mu > 0$ and scale parameter $\lambda > 0$. Its density is

$$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp^{\frac{-\lambda(x-\mu)^2}{2\mu^2 x}}, \qquad \text{for } x > 0. \tag{43}$$

The distribution function is given by

$$F(x) = \Phi\left(\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} - 1\right)\right) + \exp^{\frac{2\lambda}{\mu}} \Phi\left(-\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} + 1\right)\right), \tag{44}$$

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 (43) 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 (44) 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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat\mu,\ \hat\lambda]$ of the inverse gaussian distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [15, page 271]:

$$\begin{aligned}\hat\mu &= \frac{1}{n}\sum_{i=1}^{n} X_i\\ \frac{1}{\hat\lambda} &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{X_i} - \frac{1}{\hat\mu}\right)\end{aligned}$$

`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.

JohnsonSBDist

Extends the class `ContinuousDistribution` for the *Johnson* S_B distribution (see [19, 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

$$f(x) = \frac{\delta}{\lambda y(1-y)\sqrt{2\pi}} \exp(-(1/2)\left[\gamma + \delta \ln\left(y/(1-y)\right)\right]^2) \text{ for } \xi < x < \xi + \lambda, \tag{45}$$

and 0 elsewhere. The distribution function is

$$F(x) = \Phi[\gamma + \delta \ln(y/(1-y))], \text{ for } \xi < x < \xi + \lambda, \tag{46}$$

where Φ is the standard normal distribution function. The inverse distribution function is

$$F^{-1}(u) = \xi + \lambda(1/(1 + e^{-v(u)})) \qquad \text{for } 0 \le u \le 1, \tag{47}$$

where

$$v(u) = [\Phi^{-1}(u) - \gamma]/\delta. \tag{48}$$

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 (45).

```
public static double cdf (double gamma, double delta,
                          double xi, double lambda, double x)
```

Computes the distribution function (46).

```
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 ().

```
public static double getMean (double gamma, double delta, double xi,
                              double lambda)
```

Computes and returns the mean of the Johnson S_B 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 of the Johnson S_B 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_B 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.

JohnsonSUDist

Extends the class `ContinuousDistribution` for the *Johnson* S_U distribution (see [19, 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

$$f(x) = \frac{\delta}{\lambda\sqrt{y^2+1}\sqrt{2\pi}} \exp\left(-(1/2)\left[\gamma + \delta \ln\left[y + \sqrt{y^2+1}\right]\right]^2\right) \qquad \text{for } -\infty < x < \infty, \tag{49}$$

and distribution function

$$F(x) = \Phi\left\{\gamma + \delta \ln\left[y + \sqrt{y^2+1}\right]\right\}, \qquad \text{for } -\infty < x < \infty, \tag{50}$$

where Φ is the standard normal distribution function. The inverse distribution function is

$$F^{-1}(u) = \xi + \lambda(e^{t(u)} - e^{-t(u)})/2, \qquad \text{for } 0 \le u < 1, \tag{51}$$

where

$$t(u) = [\Phi^{-1}(u) - \gamma]/\delta. \tag{52}$$

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 $\mathrm{Var}[X] = \frac{1}{2}[(e^{1/\delta^2} - 1)(e^{1/\delta^2} \cosh(2\frac{\gamma}{\delta}) + 1)]$ 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.

LaplaceDist

Extends the class `ContinuousDistribution` for the *Laplace* distribution (see, e.g., [16, page 165]). It has location parameter θ and scale parameter $\phi > 0$. The density function is given by

$$f(x) = e^{-|x-\theta|/\phi}/(2\phi) \qquad \text{for } -\infty < x < \infty. \tag{53}$$

The distribution function is

$$F(x) = \begin{cases} \frac{1}{2}e^{(x-\theta)/\phi} & \text{if } x \le \theta, \\ 1 - \frac{1}{2}e^{(\theta-x)/\phi} & \text{otherwise,} \end{cases} \tag{54}$$

and its inverse is

$$F^{-1}(u) = \begin{cases} \phi\log(2u) + \theta & \text{if } 0 \le u \le \frac{1}{2}, \\ \theta - \phi\log(2(1-u)) & \text{otherwise.} \end{cases} \tag{55}$$

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

import umontreal.iro.lecuyer.util.Num;

public class LaplaceDist extends ContinuousDistribution
```

Constructors

`public LaplaceDist()`

Constructs a `LaplaceDist` object with default parameters $\theta = 0$ and $\phi = 1$.

`public LaplaceDist (double theta, double phi)`

Constructs a `LaplaceDist` object with parameters $\theta =$ `theta` and $\phi =$ `phi`.

Methods

`public static double density (double theta, double phi, double x)`

Computes the Laplace density function.

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

Computes the Laplace distribution function.

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

Computes the Laplace complementary distribution function.

`public static double inverseF (double theta, double phi, double u)`

Computes the inverse Laplace distribution function.

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

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

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

Estimates and returns the parameters $[\hat{\theta}, \hat{\phi}]$ of the Laplace distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [16, page 172]:

$$\begin{aligned}\hat{\theta} &= \text{the median of } \{X_1, \ldots, X_n\} \\ \hat{\phi} &= \frac{1}{n}\sum_{i=1}^{n} |X_i - \hat{\theta}|\end{aligned}$$

`public static double getMean (double theta, double phi)`

Computes and returns the mean $E[X] = \theta$ of the Laplace distribution with parameters θ and ϕ.

`public static double getVariance (double theta, double phi)`

Computes and returns the variance $\mathrm{Var}[X] = 2\phi^2$ of the Laplace distribution with parameters θ and ϕ.

`public static double getStandardDeviation (double theta, double phi)`

Computes and returns the standard deviation of the Laplace distribution with parameters θ and ϕ.

`public double getTheta()`

Returns the parameter θ.

`public double getPhi()`

Returns the parameter ϕ.

LogisticDist

Extends the class `ContinuousDistribution` for the *logistic* distribution (e.g., [16, page 115]). It has location parameter α and scale parameter $\lambda > 0$. The density is

$$f(x) = \frac{\lambda e^{-\lambda(x-\alpha)}}{(1 + e^{-\lambda(x-\alpha)})^2} \qquad \text{for } -\infty < x < \infty, \tag{56}$$

and the distribution function is

$$F(x) = \frac{1}{1 + e^{-\lambda(x-\alpha)}} \qquad \text{for } -\infty < x < \infty. \tag{57}$$

For $\lambda = 1$ and $\alpha = 0$, one can write

$$F(x) = \frac{1 + \tanh(x/2)}{2}. \tag{58}$$

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 $\alpha =$ `alpha` and $\lambda =$ `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 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 in table $x[i]$, $i = 0, 1, \dots, n-1$.

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

Estimates and returns the parameters $[\hat{\alpha}, \hat{\lambda}]$ of the log-normal distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \dots, n-1$.

The equations of the maximum likelihood are defined in [8, page 128]:

$$\begin{aligned}\sum_{i=1}^{n} \frac{1}{1+e^{\hat{\lambda}(X_i-\hat{\alpha})}} &= \frac{n}{2} \\ \sum_{i=1}^{n} \hat{\lambda}(X_i-\hat{\alpha}) \frac{1-e^{\hat{\lambda}(X_i-\hat{\alpha})}}{1+e^{\hat{\lambda}(X_i-\hat{\alpha})}} &= n\end{aligned}$$

`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.

LoglogisticDist

Extends the class `ContinuousDistribution` for the *Log-Logistic* distribution with shape parameter $\alpha > 0$ and scale parameter $\beta > 0$. Its density is

$$f(x) = \frac{\alpha (x/\beta)^{\alpha-1}}{\beta [1 + (x/\beta)^{\alpha}]^2} \qquad \text{for } x > 0 \tag{59}$$

and its distribution function is

$$F(x) = \frac{1}{1 + (\frac{x}{\beta})^{-\alpha}} \qquad \text{for } x > 0. \tag{60}$$

The complementary distribution is

$$\bar{F}(x) = \frac{1}{1 + (\frac{x}{\beta})^{\alpha}} \qquad \text{for } x > 0. \tag{61}$$

```
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 (59) for a log-logisitic distribution with parameters α and β.

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

Computes the distribution function (60) 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 (61) 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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{\alpha}, \hat{\beta}]$ of the log-logistic distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The estimate of the parameters is given by maximizing numerically the log-likelihood function, using the Uncmin package [26, 28].

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

Computes and returns the mean $E[X] = \beta\theta\,\mathrm{cosec}(\theta)$, 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.

LognormalDist

Extends the class `ContinuousDistribution` for the *lognormal* distribution [15]. It has scale parameter μ and shape parameter $\sigma > 0$. The density is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\ln(x)-\mu)^2/(2\sigma^2)} \qquad \text{for } x > 0, \tag{62}$$

and 0 elsewhere. The distribution function is

$$F(x) = \Phi\left((\ln(x) - \mu)/\sigma\right) \qquad \text{for } x > 0, \tag{63}$$

where Φ is the standard normal distribution function. Its inverse is given by

$$F^{-1}(u) = e^{\mu + \sigma\Phi^{-1}(u)} \qquad \text{for } 0 \le u < 1. \tag{64}$$

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 (62).

```
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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters [$\hat{\mu}$, $\hat{\sigma}$] of the log-normal distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [15, page 220]:

$$\begin{aligned}\hat{\mu} &= \frac{1}{n}\sum_{i=1}^{n}\ln(X_i)\\ \hat{\sigma} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\ln(X_i)-\hat{\mu})^2}\end{aligned}$$

`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.

NormalDist

Extends the class `ContinuousDistribution` for the *normal* distribution (e.g., [15, page 80]). It has mean μ and variance σ^2. Its density function is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)} \qquad \text{for } -\infty < x < \infty, \tag{65}$$

where $\sigma > 0$. When $\mu = 0$ and $\sigma = 1$, we have the *standard normal* distribution, with corresponding distribution function

$$F(x) = \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt \qquad \text{for } -\infty < x < \infty. \tag{66}$$

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 parameters $\mu =$ `mu` and $\sigma =$ `sigma`.

Methods

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

Computes the normal density function (65).

`public static double cdf01 (double x)`

Same as `cdf (0.0, 1.0, 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 [27], which gives 16 decimals of precision.

`public static double barF01 (double x)`

Same as `barF (0.0, 1.0, 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. Uses a Chebyshev series giving 16 decimal digits of precision [27].

`public static double inverseF01 (double u)`

Same as `inverseF (0.0, 1.0, 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 rational Chebyshev approximations giving at least 16 decimal digits of precision (see [4]).

`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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{\mu}, \hat{\sigma}]$ of the normal distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [15, page 123]:

$$\begin{aligned}\hat{\mu} &= \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i\\ \hat{\sigma} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2}\end{aligned}$$

`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.

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 parameters $\mu =$ `mu` and $\sigma =$ `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 [22] 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 [22] 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 [22], with table lookups. Returns 6 decimal digits of precision. This method is approximately 20% faster than `NormalDist.inverseF`.

ParetoDist

Extends the class `ContinuousDistribution` for a distribution from the *Pareto* family, with shape parameter $\alpha > 0$ and location parameter $\beta > 0$ [15, page 574]. The density for this type of Pareto distribution is

$$f(x) = \frac{\alpha\beta^\alpha}{x^{\alpha+1}} \qquad \text{for } x \ge \beta, \tag{67}$$

and 0 otherwise. The distribution function is

$$F(x) = 1 - (\beta/x)^\alpha \qquad \text{for } x \ge \beta, \tag{68}$$

and the inverse distribution function is

$$F^{-1}(u) = \beta(1-u)^{-1/\alpha} \qquad \text{for } 0 \le u < 1.$$

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

public class ParetoDist extends ContinuousDistribution
```

Constructors

`public ParetoDist (double alpha)`

Constructs a `ParetoDist` object with parameters α = `alpha` and $\beta = 1$.

`public ParetoDist (double alpha, double beta)`

Constructs a `ParetoDist` object with parameters α = `alpha` and β = `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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{\alpha}, \hat{\beta}]$ of the Pareto distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [15, page 581]:

$$\begin{aligned} \hat{\beta} &= \min_i \{X_i\} \\ \hat{\alpha} &= \frac{n}{\sum_{i=1}^{n} \ln\left(\frac{X_i}{\hat{\beta}}\right)} \end{aligned}$$

`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.

Pearson5Dist

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

$$f(x) = \begin{cases} \dfrac{x^{-(\alpha+1)}e^{-\beta/x}}{\beta^{-\alpha}\Gamma(\alpha)} & \text{for } x > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{69}$$

where Γ is the gamma function. The distribution function is given by

$$F(x) = 1 - F_G\left(\frac{1}{x}\right) \qquad \text{for } x > 0, \tag{70}$$

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 Pearson5Dist extends ContinuousDistribution
```

Constructor

`public Pearson5Dist (double alpha, double beta)`

Constructs a `Pearson5Dist` object with parameters $\alpha =$ `alpha` and $\beta =$ `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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{\alpha}, \hat{\beta}]$ of the Pearson V distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are the same as the equations of the gamma distribution, with the sample $Y_i = \frac{1}{X_i}$.

`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.

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

$$f(x) = \begin{cases} \dfrac{(x/\beta)^{\alpha_1 - 1}}{\beta \mathcal{B}(\alpha_1, \alpha_2)(1 + x/\beta)^{\alpha_1 + \alpha_2}} & \text{for } x > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{71}$$

where $\mathcal{B}$ is the beta function. The distribution function is given by

$$F(x) = F_B\left(\frac{x}{x + \beta}\right) \qquad \text{for } x > 0, \tag{72}$$

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 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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{\alpha_1}, \hat{\alpha_2}, \hat{\beta}]$ of the Pearson VI distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The estimate of the parameters is given by maximizing numerically the log-likelihood function, using the Uncmin package [26, 28].

```
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.

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., [19, page 318]). The distribution function starts at $X_{(1)}$ and climbs linearly by $1/(n-1)$ between any two successive observations. The density is

$$f(x) = \frac{1}{(n-1)(X_{(i+1)} - X_{(i)})} \text{ for } X_{(i)} \le x < X_{(i+1)} \text{ and } i = 1, 2, \dots, n-1. \tag{73}$$

The distribution function is

$$F(x) = \begin{cases} 0 & \text{for } x < X_{(1)}, \\ \dfrac{i-1}{n-1} + \dfrac{x - X_{(i)}}{(n-1)(X_{(i+1)} - X_{(i)})} & \text{for } X_{(i)} \le x < X_{(i+1)} \text{ and } i < n, \\ 1 & \text{for } x \ge X_{(n)}, \end{cases} \tag{74}$$

whose inverse is

$$F^{-1}(u) = X_{(i)} + ((n-1)u - i + 1)(X_{(i+1)} - X_{(i)}) \tag{75}$$

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.

StudentDist

Extends the class `ContinuousDistribution` for the *Student-t* distribution [16, page 362] with n degrees of freedom, where n is a positive integer. Its density is

$$f(x) = \frac{\Gamma\left((n+1)/2\right)}{\Gamma(n/2)\sqrt{\pi n}} \left(1 + \frac{x^2}{n}\right)^{-(n+1)/2} \qquad \text{for } -\infty < x < \infty, \tag{76}$$

where $\Gamma(x)$ is the gamma function defined in (40).

The non-static methods `cdf` and `barF` use the same algorithm as in `cdf`.

```
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 (76) for a Student-t distribution with n degrees of freedom.

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

Returns the approximation of [18, page 96] for the Student-t distribution function with n degrees of freedom. Gives at least 12 decimals of precision for $n \le 10^3$, and at least 10 decimals for $10^3 < n \le 10^5$.

`public static double cdf2 (int n, int d, double x)`

Returns an approximation of the Student-t distribution function with n degrees of freedom. Uses the relationship (see [15])

$$2F(x) = \begin{cases} I_{n/2,1/2}(n/(n+x^2)) & \text{for } x < 0, \\ I_{1/2,n/2}(x^2/(n+x^2)) & \text{for } x \ge 0, \end{cases} \tag{77}$$

where $I_{\alpha,\beta}$ is the beta distribution function with parameters α and β (also called the incomplete *beta* ratio) defined in (23), which is approximated by calling `BetaDist.cdf`. The function tries to return d decimals digits of precision (but there is no guarantee). This method is much slower (twenty to forty times, depending on parameters) than `cdf`, but could be used if precision is important.

`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. Uses an approximation giving at least 5 decimal digits of precision when $n \geq 8$ or $n \leq 2$, and 3 decimal digits of precision when $3 \leq n \leq 7$ (see [13] and Figure L.28 of [6]).

`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 in table $x[i]$, $i = 0, 1, \ldots, m-1$.

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

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

`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.

TriangularDist

Extends the class `ContinuousDistribution` for the *triangular* distribution (see [16, page 297] and [19, page 317]) with domain $[a, b]$ and *mode* (or shape parameter) m, where $a \le m \le b$. The density function is

$$f(x) = \begin{cases} \frac{2(x-a)}{(b-a)(m-a)} & \text{if } a \le x \le m, \\ \frac{2(b-x)}{(b-a)(b-m)} & \text{if } m \le x \le b, \\ 0 & \text{elsewhere,} \end{cases} \tag{78}$$

the distribution function is

$$F(x) = \begin{cases} 0 & \text{for } x < a, \\ \frac{(x-a)^2}{(b-a)(m-a)} & \text{if } a \le x \le m, \\ 1 - \frac{(b-x)^2}{(b-a)(b-m)} & \text{if } m \le x \le b, \\ 1 & \text{for } x > b, \end{cases} \tag{79}$$

and the inverse distribution function is given by

$$F^{-1}(u) = \begin{cases} a + \sqrt{(b-a)(m-a)u} & \text{if } 0 \le u \le \frac{m-a}{b-a}, \\ b - \sqrt{(b-a)(b-m)(1-u)} & \text{if } \frac{m-a}{b-a} \le u \le 1. \end{cases} \tag{80}$$

```
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 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.

UniformDist

Extends the class `ContinuousDistribution` for the *uniform* distribution [16, page 276] over the interval $[a, b]$. Its density is

$$f(x) = 1/(b-a) \qquad \text{for } a \le x \le b \tag{81}$$

and 0 elsewhere. The distribution function is

$$F(x) = (x-a)/(b-a) \qquad \text{for } a \le x \le b \tag{82}$$

and its inverse is

$$F^{-1}(u) = a + (b-a)u \qquad \text{for } 0 \le u \le 1. \tag{83}$$

```
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 (81).

`public static double cdf (double a, double b, double x)`

Computes the uniform distribution function as in (82).

`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 (83).

`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 in table $x[i]$, $i = 0, 1, \ldots, n-1$.

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

Estimates and returns the parameters $[\hat{a}, \hat{b}]$ of the uniform distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [19, page 300]:

$$\begin{aligned} \hat{a} &= \min_i \{X_i\} \\ \hat{b} &= \max_i \{X_i\} \end{aligned}$$

`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 $\text{Var}[X] = (b - a)^2/12$ 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.

WeibullDist

This class extends the class `ContinuousDistribution` for the *Weibull* distribution [15, page 628] with shape parameter $\alpha > 0$, location parameter δ, and scale parameter $\lambda > 0$. The density function is

$$f(x) = \alpha\lambda^{\alpha}(x-\delta)^{\alpha-1}e^{-(\lambda(x-\delta))^{\alpha}} \qquad \text{for } x > \delta, \tag{84}$$

the distribution function is

$$F(x) = 1 - e^{-(\lambda(x-\delta))^{\alpha}} \qquad \text{for } x > \delta, \tag{85}$$

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, 0.0, 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, 0.0, 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, 0.0, 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, 0.0, x)`.

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

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

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

Estimates and returns the parameters $[\hat{\alpha},\ \hat{\lambda},\ \hat{\delta} = 0]$ of the Weibull distribution using the maximum likelihood method based on the n observations in table $x[i]$, $i = 0, 1, \ldots, n-1$.

The equations of the maximum likelihood are defined in [19, page 303]:

$$\begin{aligned} \frac{\sum_{i=1}^{n} X_i^{\hat{\alpha}} \ln(X_i)}{\sum_{i=1}^{n} X_i^{\hat{\alpha}}} - \frac{1}{\hat{\alpha}} &= \frac{\sum_{i=1}^{n} \ln(X_i)}{n} \\ \hat{\lambda} &= \left(\frac{n}{\sum_{i=1}^{n} X_i^{\hat{\alpha}}} \right)^{1/\hat{\alpha}} \end{aligned}$$

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

Computes and returns the mean $E[X] = \delta + \frac{\Gamma(1+\frac{1}{\alpha})}{\lambda}$ 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 $\mathrm{Var}[X] = \frac{1}{\lambda^2}|\Gamma(\frac{2}{\alpha}+1) - \Gamma^2(\frac{1}{\alpha}+1)|$ 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.

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 and F, the new ones are f^* and F^*, defined by

$$f^*(x) = f(x)/(F(b) - F(a)) \qquad \text{for } a \le x \le b$$

and $f^*(x) = 0$ and zero elsewhere, and

$$F^*(x) = F(x)/(F(b) - F(a)) \qquad \text{for } a \le x \le b.$$

The inverse distribution function of the truncated distribution is

$$F^{-1^*}(u) = F^{-1}(F(a) + (F(b) - F(a))u)$$

where F^{-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 $[a, b]$. If `a = Double.NEGATIVE_INFINITY`, $F(a)$ is assumed to be 0. If `b = Double.POSITIVE_INFINITY`, $F(b)$ is assumed to be 1.

Methods

`public double getA()`

Returns the value of a.

`public double getB()`

Returns the value of b.

`public double getFa()`

Returns the value of $F(a)$.

`public double getFb()`

Returns the value of $F(b)$.

`public double getArea()`

Returns the value of $F(b) - F(a)$, 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.

References

[1] D. J. Best and D. E. Roberts. Algorithm AS 91: The percentage points of the χ^2 distribution. *Applied Statistics*, 24:385–388, 1975.

[2] G. P. Bhattacharjee. The incomplete gamma integral. *Applied Statistics*, 19:285–287, 1970. AS32.

[3] Z. W. Birnbaum and S. C. Saunders. A new family of life distributions. *Journal of Applied Probability*, 6:319–327, 1969.

[4] 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.

[5] L. N. Bol'shev. Some applications of Pearson transformations. *Review of the Internat. Stat. Institute*, 32:14–16, 1964.

[6] P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation.* Springer-Verlag, New York, second edition, 1987.

[7] B. H. Camp. Approximation to the point binomial. *Ann. Math. Stat.*, 22:130–131, 1951.

[8] M. Evans and T. Swartz. *Approximating Integrals via Monte Carlo and Deterministic Methods.* Oxford University Press, Oxford, UK, 2000.

[9] W. Gautschi. Algorithm 222: Incomplete beta function ratios. *Communications of the ACM*, 7(3):143–144, 1964.

[10] W. Gautschi. Certification of algorithm 222: Incomplete beta function ratios. *Communications of the ACM*, 7(3):244, 1964.

[11] J. E. Gentle. *Random Number Generation and Monte Carlo Methods.* Springer, New York, 1998.

[12] R. B. Goldstein. Algorithm 451: Chi-square quantiles. *Communications of the ACM*, 16:483–485, 1973.

[13] G. W. Hill. Algorithm 395: Student's t-distribution. *Communications of the ACM*, 13:617–619, 1970.

[14] N. L. Johnson and S. Kotz. *Distributions in Statistics: Discrete Distributions.* Houghton Mifflin, Boston, 1969.

[15] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume 1. Wiley, 2nd edition, 1994.

[16] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume 2. Wiley, 2nd edition, 1995.

[17] V. Kachitvichyanukul and B. Schmeiser. Computer generation of hypergeometric random variates. *J. Statist. Comput. Simul.*, 22:127–145, 1985.

[18] W. J. Kennedy Jr. and J. E. Gentle. *Statistical Computing.* Dekker, New York, 1980.

[19] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis.* McGraw-Hill, New York, third edition, 2000.

[20] J. Leydold and W. Hörmann. *UNURAN—A Library for Universal Non-Uniform Random Number Generators*, 2002. Available at `http://statistik.wu-wien.ac.at/unuran`.

[21] K. V. Mardia and P. J. Zemroch. *Tables of the F and Related Distributions with Algorithms.* Academic Press, London, 1978.

[22] G. Marsaglia, A. Zaman, and J. C. W. Marsaglia. Rapid evaluation of the inverse normal distribution function. *Statistic and Probability Letters*, 19:259–266, 1994.

[23] W. Molenaar. *Approximations to the Poisson, Binomial and Hypergeometric Distribution Functions*, volume 31 of *Mathematical Center Tract.* Mathematisch Centrum, Amsterdam, 1970.

[24] S. L. Moshier. Cephes math library, 2000. See `http://www.moshier.net`.

[25] 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.

[26] R. B. Schnabel. *UNCMIN—Unconstrained Optimization Package, FORTRAN.* University of Colorado at Boulder. See `http://www.ici.ro/camo/unconstr/uncmin.htm`.

[27] J. L. Schonfelder. Chebyshev expansions for the error and related functions. *Mathematics of Computation*, 32:1232–1240, 1978.

[28] S. P. Verrill. *UNCMIN—Unconstrained Optimization Package, Java.* US Forest Service, Forest Products Laboratory. Available at `http://www1.fpl.fs.fed.us/optimization.html`.