

SSJ User's Guide

Package `probdist`

Probability Distributions

Version: January 26, 2006

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	2
General Classes	4
DiscreteDistributionInt	4
Distribution	5
DiscreteDistribution	6
ContinuousDistribution	7
DistributionFactory	8
ContinuousDistribution2Dim	9
Discrete Distributions over Integers	10
BinomialDist	10
GeometricDist	12
HypergeometricDist	13
LogarithmicDist	15
NegativeBinomialDist	16
PoissonDist	17
Discrete Distributions over Real Numbers	19
EmpiricalDist	19
Continuous Distributions	20
BetaDist	20
BetaSymmetricalDist	23
BiNormalDist	24
BiNormalGenzDist	26
BiNormalDonnellyDist	27
BivariateTDist	28
CauchyDist	29
ChiDist	30
ChiSquareDist	31
ChiSquareDistQuick	33
ErlangDist	34

ExponentialDist	35
ExtremeValueDist	36
GammaDist	37
JohnsonSBDist	39
JohnsonSUDist	41
LaplaceDist	43
LogisticDist	44
LognormalDist	46
NormalDist	48
NormalDistQuick	50
ParetoDist	51
PiecewiseLinearEmpiricalDist	53
StudentDist	55
TriangularDist	57
UniformDist	59
WeibullDist	60
TruncatedDist	62

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

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

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

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

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

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

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

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

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

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 $\bar{F}(x) = P[X \geq x]$ 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

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

for $0 \leq u \leq 1$. 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 $F(x) \leq 1/2$ 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 of the abstract class `Distribution` (and sometimes `ContinuousDistribution`) are required by some methods in the classes `GofStat` and `GofFormat`, in package `gof`.

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
```

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

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

```
    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 int inverseF (double u)
```

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

Distribution

This interface should be implemented by classes supporting discrete and continuous distributions over the *real numbers*. It specifies the signatures of methods that compute the distribution function $F(x)$, the complementary distribution function $\bar{F}(x)$, and the inverse distribution function $F^{-1}(u)$.

For distributions over integers, use the class `DiscreteDistributionInt`.

```
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).

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 $x_0 < \dots < x_{n-1}$ (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 $[0, 1]$.

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 [18].

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 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. 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?

ContinuousDistribution2Dim

Classes implementing 2-dimensionnal continuous distributions should inherit from this class. Such distributions are characterized by a *density* function $f(x, y)$, thus the signature of a `density` method is supplied here. This class also provides a default implementation of $\bar{F}(x, y)$, the upper CDF. The inverse function $F^{-1}(u)$ represents a curve $y = h(x)$ of constant u . Thus it is not implemented here. 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 ContinuousDistribution2Dim
{
    public int decPrec = 15;

    public abstract double density (double x, double y);
    Returns  $f(x, y)$ , the density of  $(X, Y)$  evaluated at  $(x, y)$ .

    public abstract double cdf (double x, double y);
```

Computes and returns the distribution function $F(x, y)$:

$$F(x, y) = P[X \leq x, Y \leq y] = \int_{-\infty}^x ds \int_{-\infty}^y dt f(s, t). \quad (6)$$

```
public double barF (double x, double y)
```

Returns the upper cumulative distribution function $\bar{F}(x, y)$:

$$\bar{F}(x, y) = P[X \geq x, Y \geq y] = \int_x^{\infty} ds \int_y^{\infty} dt f(s, t). \quad (7)$$

The default implementation computes $\bar{F}(x, y) = 1 + F(x, y) - F(\infty, y) - F(x, \infty)$.

BinomialDist

Extends the class `DiscreteDistributionInt` for the *binomial* distribution [17, page 321] with parameters n and p , where n is a positive integer and $0 \leq p \leq 1$. Its mass function is given by

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

and its distribution function is

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

```
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 (8), for $0 \leq x \leq n$, 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

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

where p and q are arbitrary real numbers (q is not necessarily equal to $1 - p$). In the case where $0 \leq p \leq 1$ 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 $n \leq 10000$, the non-negligible terms of the sum are added explicitly. If $n > 10000$ and $np(1 - p) > 100$, the Camp-Paulson normal approximation [4, 21] is used, otherwise, a Poisson approximation due to Bol'shev [2, 21] 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 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 [17, page 322] with parameter p , where $0 < p < 1$. Its mass function is

$$p(x) = p(1 - p)^x, \quad \text{for } x = 0, 1, 2, \dots \quad (11)$$

The distribution function is given by

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

and its inverse is

$$F^{-1}(u) = \left\lceil \frac{\ln(1 - u)}{\ln(1 - p)} \right\rceil, \quad \text{for } 0 \leq u < 1. \quad (13)$$

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

```
public class GeometricDist extends DiscreteDistributionInt
```

Constructor

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

Constructs a geometric distribution with parameter p .

Methods

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

Computes the probability mass function $p(x)$ given by (11) .

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

Computes the distribution function $F(x)$.

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

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

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

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

```
    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 [8, 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 $1 \leq m \leq l$ and $1 \leq k \leq l$. Its mass function is given by

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

```
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 (14).

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

Computes the distribution function $F(x)$.

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

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

```
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 [15].

```
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

$$p(x) = \frac{-1}{\log(1-\theta)} \frac{\theta^x}{x} \quad \text{for } x = 1, 2, \dots \quad (15)$$

Its distribution function is

$$F(x) = \begin{cases} 0, & \text{for } x \leq 0. \\ \frac{-1}{\log(1-\theta)} \sum_{i=1}^x \frac{\theta^i}{i}, & \text{for } x > 0. \end{cases} \quad (16)$$

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

```
public class LogarithmicDist extends DiscreteDistributionInt
```

Constructor

```
    public LogarithmicDist (double theta)
```

Constructs a logarithmic distribution with parameter θ .

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 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 [17, page 324] with parameters n and p , where n is a positive integer and $0 \leq p \leq 1$. Its mass function is

$$p(x) = \binom{n+x-1}{x} p^n (1-p)^x, \quad \text{for } x = 0, 1, 2, \dots \quad (17)$$

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

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

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

Methods

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

Computes the probability mass function defined in (17).

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

Computes the distribution function.

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

Sets the parameter n and p of this object.

PoissonDist

Extends the class `DiscreteDistributionInt` for the *Poisson* distribution [17, page 325] with mean $\lambda \geq 0$. The mass function is

$$p(x) = \frac{e^{-\lambda} \lambda^x}{x!}, \quad \text{for } x = 0, 1, \dots \quad (18)$$

and the distribution function is

$$F(x) = e^{-\lambda} \sum_{j=0}^x \frac{\lambda^j}{j!}, \quad \text{for } x = 0, 1, \dots \quad (19)$$

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 $\lambda = \text{lambda}$. If $\lambda \geq 20$, this (static) method uses the logarithm of the gamma function, defined in (42), 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 $\lambda = \text{lambda}$.

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 $\lambda = \text{lambda}$. 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 approximation (see the static `cdf` method) and calls `GammaDist.inverseF(x+1, 15, lambda)`.

```
public double getLambda()
```

Returns the λ associated with this object.

```
public void setLambda (double lambda)
```

Sets the λ associated with this object.

EmpiricalDist

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

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

```
public class EmpiricalDist extends DiscreteDistribution
```

```
    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 (InputStreamReader in) throws IOException
```

Constructs a new empirical distribution using the observations read from the input stream 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.

```
    public int getN()
```

Returns n , the number of observations.

```
    public double getObs (int i)
```

Returns the value of $X_{(i)}$.

```
    public double getSampleMean()
```

Returns the sample mean of the observations.

```
    public double getSampleVariance()
```

Returns the sample variance of the observations.

```
    public double getSampleStandardDeviation()
```

Returns the sample standard deviation of the observations.

```
    public double getInterQuartileRange()
```

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

²The method `java.util.Arrays.sort` may be used to sort the observations.

BetaDist

Extends the class `ContinuousDistribution` for the *beta* distribution [14, page 210] with shape parameters $\alpha > 0$ and $\beta > 0$, over the interval (a, b) , where $a < b$. This distribution has density

$$f(x) = \frac{(x-a)^{\alpha-1}(b-x)^{\beta-1}}{\mathcal{B}(\alpha, \beta)(b-a)^{\alpha+\beta-1}}, \quad \text{for } a < x < b, \text{ and } 0 \text{ elsewhere,} \quad (20)$$

and distribution function

$$F(x) = I_{\alpha, \beta}(x) = \int_a^x \frac{(\xi-a)^{\alpha-1}(b-\xi)^{\beta-1}}{\mathcal{B}(\alpha, \beta)(b-a)^{\alpha+\beta-1}} d\xi, \quad \text{for } a < x < b, \quad (21)$$

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

$$\mathcal{B}(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)}, \quad (22)$$

and $\Gamma(x)$ is the gamma function defined in (42).

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

```
public class BetaDist extends ContinuousDistribution
```

Constructors

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

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

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

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

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

Constructs a `BetaDist` object with parameters $\alpha = \text{alpha}$ and $\beta = \text{beta}$, 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 $\alpha = \text{alpha}$ and $\beta = \text{beta}$, 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 [6, 7]. Otherwise, if $\min(\alpha, \beta) \leq 30$, uses an approximation due to Bol'shev [19], else uses a normal approximation [23]. The method tries its best to return *d* decimals digits of precision. For $d \leq 13$, 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 [22]. 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 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 $\alpha = \beta = \text{alpha}$, 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 $0 < \alpha = \beta = \text{alpha}$. 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 [23].

BiNormalDist

Extends the class `ContinuousDistribution2Dim` for the *bivariate normal* distribution [12, page 84]. It has means $E[X] = \mu_1$, $E[Y] = \mu_2$, and variances $\text{var}[X] = \sigma_1^2$, $\text{var}[Y] = \sigma_2^2$ such that $\sigma_1 > 0$ and $\sigma_2 > 0$. The correlation between X and Y is ρ . Its density function is

$$f(x, y) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} e^{-T} \quad (23)$$

$$T = \frac{1}{2(1-\rho^2)} \left[\left(\frac{x-\mu_1}{\sigma_1} \right)^2 - 2\rho \left(\frac{x-\mu_1}{\sigma_1} \right) \left(\frac{y-\mu_2}{\sigma_2} \right) + \left(\frac{y-\mu_2}{\sigma_2} \right)^2 \right]$$

and the corresponding distribution function is (the `cdf` method)

$$\Phi(\mu_1, \sigma_1, x, \mu_2, \sigma_2, y, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \int_{-\infty}^x dx \int_{-\infty}^y dy e^{-T}. \quad (24)$$

We also define the upper distribution function (the `barF` method) as

$$\bar{\Phi}(\mu_1, \sigma_1, x, \mu_2, \sigma_2, y, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \int_x^{\infty} dx \int_y^{\infty} dy e^{-T}. \quad (25)$$

When $\mu_1 = \mu_2 = 0$ and $\sigma_1 = \sigma_2 = 1$, we have the *standard binormal* distribution, with corresponding distribution function

$$\Phi(x, y, \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^x dx \int_{-\infty}^y dy e^{-S} \quad (26)$$

$$S = \frac{x^2 - 2\rho xy + y^2}{2(1-\rho^2)}.$$

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

```
public class BiNormalDist extends ContinuousDistribution2Dim
```

Constructors

```
public BiNormalDist (double rho)
```

Constructs a `BiNormalDist` object with default parameters $\mu_1 = \mu_2 = 0$, $\sigma_1 = \sigma_2 = 1$ and correlation $\rho = \text{rho}$.

```
public BiNormalDist (double mu1, double sigma1,
                    double mu2, double sigma2, double rho)
```

Constructs a `BiNormalDist` object with parameters $\mu_1 = \text{mu1}$, $\mu_2 = \text{mu2}$, $\sigma_1 = \text{sigma1}$, $\sigma_2 = \text{sigma2}$ and $\rho = \text{rho}$.

Methods

`public static double density (double x, double y, double rho)`

Computes the *standard binormal* density function (23) with $\mu_1 = \mu_2 = 0$ and $\sigma_1 = \sigma_2 = 1$.

`public static double density (double mu1, double sigma1, double x,
double mu2, double sigma2, double y,
double rho)`

Computes the *binormal* density function (23) with parameters $\mu_1 = \text{mu1}$, $\mu_2 = \text{mu2}$, $\sigma_1 = \text{sigma1}$, $\sigma_2 = \text{sigma2}$ and $\rho = \text{rho}$.

`public static double cdf (double x, double y, double rho)`

Computes the standard *binormal* distribution (26) with $\mu_1 = \mu_2 = 0$ and $\sigma_1 = \sigma_2 = 1$. The simple algorithm used here is slower and less precise than the `cdf` in both `BiNormalGenzDist` and in `BiNormalDonnellyDist`.

`public static double cdf (double mu1, double sigma1, double x,
double mu2, double sigma2, double y,
double rho)`

Computes the *binormal* distribution function (24) with parameters $\mu_1 = \text{mu1}$, $\mu_2 = \text{mu2}$, $\sigma_1 = \text{sigma1}$, $\sigma_2 = \text{sigma2}$ and $\rho = \text{rho}$.

`public static double barF (double x, double y, double rho)`

Computes the standard upper *binormal* distribution with $\mu_1 = \mu_2 = 0$ and $\sigma_1 = \sigma_2 = 1$.

`public static double barF (double mu1, double sigma1, double x,
double mu2, double sigma2, double y,
double rho)`

Computes the upper *binormal* distribution function (25) with parameters $\mu_1 = \text{mu1}$, $\mu_2 = \text{mu2}$, $\sigma_1 = \text{sigma1}$, $\sigma_2 = \text{sigma2}$ and $\rho = \text{rho}$.

`public double getMu1()`

Returns the parameter μ_1 .

`public double getMu2()`

Returns the parameter μ_2 .

`public double getSigma1()`

Returns the parameter σ_1 .

`public double getSigma2()`

Returns the parameter σ_2 .

`protected void setParams (double mu1, double sigma1,
double mu2, double sigma2, double rho)`

Sets the parameters $\mu_1 = \text{mu1}$, $\mu_2 = \text{mu2}$, $\sigma_1 = \text{sigma1}$, $\sigma_2 = \text{sigma2}$ and $\rho = \text{rho}$ of this object.

BiNormalGenzDist

Extends the class `BiNormalDist` for the *bivariate normal* distribution [12, page 84] using Genz's algorithm as described in [9].

```
package umontreal.iro.lecuyer.probdist;  
  
public class BiNormalGenzDist extends BiNormalDist
```

Constructors

```
public BiNormalGenzDist (double rho)
```

Constructs a `BiNormalGenzDist` object with default parameters $\mu_1 = \mu_2 = 0$, $\sigma_1 = \sigma_2 = 1$ and correlation $\rho = \text{rho}$.

```
public BiNormalGenzDist (double mu1, double sigma1,  
                        double mu2, double sigma2, double rho)
```

Constructs a `BiNormalGenzDist` object with parameters $\mu_1 = \text{mu1}$, $\mu_2 = \text{mu2}$, $\sigma_1 = \text{sigma1}$, $\sigma_2 = \text{sigma2}$ and $\rho = \text{rho}$.

Methods

```
public static double cdf (double x, double y, double rho)
```

Computes the standard *binormal* distribution (26) with the method described in [9]. The code for the `cdf` was translated directly from the Matlab code written by Alan Genz and available from his web page at <http://www.math.wsu.edu/faculty/genz/homepage> (the code is copyrighted by Alan Genz and is included in this package with the kind permission of the author). The absolute error is expected to be smaller than $0.5 \cdot 10^{-15}$.

BiNormalDonnellyDist

Extends the class BiNormalDist for the *bivariate normal* distribution [12, page 84] using a translation of Donnelly's FORTRAN code in [5].

```
package umontreal.iro.lecuyer.probdist;  
  
public class BiNormalDonnellyDist extends BiNormalDist
```

Constructors

```
public BiNormalDonnellyDist (double rho)
```

Constructs a BiNormalDonnellyDist object with default parameters $\mu_1 = \mu_2 = 0$, $\sigma_1 = \sigma_2 = 1$ and correlation $\rho = \text{rho}$.

```
public BiNormalDonnellyDist (double mu1, double sigma1,  
                             double mu2, double sigma2, double rho)
```

Constructs a BiNormalDonnellyDist object with parameters $\mu_1 = \text{mu1}$, $\mu_2 = \text{mu2}$, $\sigma_1 = \text{sigma1}$, $\sigma_2 = \text{sigma2}$ and $\rho = \text{rho}$.

Methods

```
public static double cdf (double x, double y, double rho)
```

Computes the standard *binormal* distribution (26) with the method described in [5]. The code was translated from the Fortran program written by T. G. Donnelly and is copyrighted by the ACM (see http://www.acm.org/pubs/copyright_policy/#Notice). The absolute error is expected to be smaller than 10^{-15} .

BivariateTDist

Extends the class `ContinuousDistribution2Dim` for the *standard bivariate t* distribution [12, page 132]. The correlation between X and Y is ρ and the number of degrees of freedom is ν . Its probability density is

$$f(x, y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \left[1 + \frac{x^2 - 2\rho xy + y^2}{\nu(1-\rho^2)} \right]^{-(\nu+2)/2}, \quad (27)$$

and the corresponding distribution function (the `cdf`) is

$$T_\nu(x, y, \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^x dx \int_{-\infty}^y dy f(x, y). \quad (28)$$

We also define the upper distribution function called `barF` as

$$\bar{T}_\nu(x, y, \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_x^\infty dx \int_y^\infty dy f(x, y). \quad (29)$$

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

public class BivariateTDist extends ContinuousDistribution2Dim
```

Constructor

```
public BivariateTDist (int nu, double rho)
    Constructs a BivariateTDist object with correlation  $\rho = \text{rho}$  and  $\nu = \text{nu}$  degrees of freedom.
```

Methods

```
public static double density (int nu, double x, double y, double rho)
    Computes the standard bivariate t density function (27) with correlation  $\rho = \text{rho}$  and  $\nu = \text{nu}$  degrees of freedom.
```

```
public static double cdf (int nu, double x, double y, double rho)
    Computes the standard bivariate t distribution (28) using the method described in [9]. The code for the cdf was translated directly from the Matlab code written by Alan Genz and available from his web page at http://www.math.wsu.edu/faculty/genz/homepage (the code is copyrighted by Alan Genz and is included in this package with the kind permission of the author). The correlation is  $\rho = \text{rho}$  and the number of degrees of freedom is  $\nu = \text{nu}$ .
```

```
public static double barF (int nu, double x, double y, double rho)
    Computes the standard upper bivariate t distribution (29).
```

```
protected void setParams (int nu, double rho)
    Sets the parameters  $\nu = \text{nu}$  and  $\rho = \text{rho}$  of this object.
```

CauchyDist

Extends the class `ContinuousDistribution` for the *Cauchy* distribution [13, page 299] with location parameter α and scale parameter $\beta > 0$. The density function is given by

$$f(x) = \frac{\beta}{\pi[(x - \alpha)^2 + \beta^2]}, \quad \text{for } -\infty < x < \infty. \quad (30)$$

The distribution function is

$$F(x) = \frac{1}{2} + \frac{\arctan((x - \alpha)/\beta)}{\pi}, \quad \text{for } -\infty < x < \infty, \quad (31)$$

and its inverse is

$$F^{-1}(u) = \alpha + \beta \tan(\pi(u - 1/2)). \quad \text{for } 0 < u < 1. \quad (32)$$

```
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  $\alpha = \text{alpha}$  and  $\beta = \text{beta}$ .
```

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 double getAlpha()
    Returns the value of  $\alpha$  for this object.

public double getBeta()
    Returns the value of  $\beta$  for this object.

public void setParams (double alpha, double beta)
    Sets the value of the parameters  $\alpha$  and  $\beta$  for this object.
```

ChiDist

Extends the class `ContinuousDistribution` for the *chi* distribution [13, page 417] with shape parameter $\nu > 0$. The density function is given by

$$f(x) = \frac{e^{-x^2/2} x^{\nu-1}}{2^{(\nu/2)-1} \Gamma(\nu/2)}, \quad \text{for } x > 0, \quad (33)$$

where $\Gamma(x)$ is the gamma function defined in (42). The distribution function is

$$F(x) = \frac{1}{\Gamma(\nu/2)} \int_0^{x^2/2} t^{\nu/2-1} e^{-t} dt. \quad (34)$$

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 an 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 int getNu()
    Returns the value of  $\nu$  for this object.

public void setNu (int nu)
    Sets the value of  $\nu$  for this object.
```

ChiSquareDist

Extends the class `ContinuousDistribution` for the *chi-square* distribution with n degrees of freedom, where n is a positive integer [13, page 416]. Its density is

$$f(x) = \frac{x^{(n/2)-1} e^{-x/2}}{2^{n/2} \Gamma(n/2)}, \quad \text{for } x > 0 \quad (35)$$

where $\Gamma(x)$ is the gamma function defined in (42). 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 (35) 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 [16, page 116] for $n \leq 350$ (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 [16, page 116] for $n \leq 350$, 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 [3]. 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 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 [3] over most of the range. For $u < 0.02$ or $u > 0.98$, it uses the approximation given in [10] for $n \geq 10$, and returns `2.0 * GammaDist.inverseF (n/2, 6, u)` for $n < 10$ in order to avoid the loss of precision of the above approximations. When $n \geq 10$ or $0.02 < u < 0.98$, it is between 20 to 30 times faster than the same method in `ChiSquareDist` for n between 10 and 1000 and even faster for larger n .

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

ErlangDist

Extends the class `ContinuousDistribution` for 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 ContinuousDistribution
```

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 $\lambda = \text{lambda}$.

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 int getK()
```

Return the parameter k for this object.

```
public double getLambda()
```

Return the parameter λ for this object.

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

Sets the parameters k , 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 [13, page 494] with mean $1/\lambda$ where $\lambda > 0$. Its density is

$$f(x) = \lambda e^{-\lambda x} \quad \text{for } x \geq 0, \quad (36)$$

its distribution function is

$$F(x) = 1 - e^{-\lambda x}, \quad \text{for } x \geq 0, \quad (37)$$

and its inverse distribution function is

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

```
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 $\lambda = \text{lambda}$.

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 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 [14, page 2], with location parameter α and scale parameter $\lambda > 0$. It has density

$$f(x) = \lambda e^{-\lambda(x-\alpha)} e^{-e^{-\lambda(x-\alpha)}}, \quad \text{for } -\infty < x < \infty, \quad (38)$$

distribution function

$$F(x) = e^{-e^{-\lambda(x-\alpha)}} \quad \text{for } -\infty < x < \infty, \quad (39)$$

and inverse distribution function

$$F^{-1}(u) = -\ln(-\ln(u))/\lambda + \alpha, \quad \text{for } 0 \leq u < 1. \quad (40)$$

```
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 $\alpha = \text{alpha}$ and $\lambda = \text{lambda}$.

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 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.

GammaDist

Extends the class `ContinuousDistribution` for the *gamma* distribution [13, page 337] with shape parameter $\alpha > 0$ and scale parameter $\lambda > 0$. The density is

$$f(x) = \frac{\lambda^\alpha x^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)}, \quad \text{for } x > 0, \quad (41)$$

where Γ is the gamma function, defined by

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx. \quad (42)$$

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 $\alpha = \text{alpha}$ and $\lambda = 1$.

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

Constructs a `GammaDist` object with parameters $\alpha = \text{alpha}$ and $\lambda = \text{lambda}$.

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

Constructs a `GammaDist` object with parameters $\alpha = \text{alpha}$ and $\lambda = \text{lambda}$, 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 $\alpha = \text{alpha}$ and $\lambda = \text{lambda}$, whose density is given by (41). The approximation is an improved version of the algorithm implemented in Fortran in [3]. The function tries to return d decimals digits of precision. For α not too large (e.g., $\alpha \leq 1000$), 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 [22]. 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 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)
```

JohnsonSBDist

Extends the class `ContinuousDistribution` for the *Johnson S_B* distribution (see [17, 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

$$f(x) = \frac{\delta}{\lambda y(1-y)\sqrt{2\pi}} \exp(-(1/2) [\gamma + \delta \ln(y/(1-y))]^2) \text{ for } \xi < x < \xi + \lambda, \quad (43)$$

and 0 elsewhere. The distribution function is

$$F(x) = \Phi[\gamma + \delta \ln(y/(1-y))], \text{ for } \xi < x < \xi + \lambda, \quad (44)$$

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

$$F^{-1}(u) = \xi + \lambda(1/(1 + e^{-v(u)})) \quad \text{for } 0 \leq u \leq 1, \quad (45)$$

where

$$v(u) = [\Phi^{-1}(u) - \gamma]/\delta. \quad (46)$$

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 (43).

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

Computes the distribution function (44).

```
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 double getGamma()  
    Returns the value of  $\gamma$  for this object.  
  
public double getDelta()  
    Returns the value of  $\delta$  for this object.  
  
public double getXi()  
    Returns the value of  $\xi$  for this object.  
  
public double getLambda()  
    Returns the value of  $\lambda$  for this object.  
  
public void setParams(double gamma, double delta, double xi, double lambda)  
    Sets the value of the parameters  $\gamma$ ,  $\delta$ ,  $\xi$  and  $\lambda$  for this object.
```

JohnsonSUDist

Extends the class `ContinuousDistribution` for the *Johnson S_U* distribution (see [17, 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

$$f(x) = \frac{\delta}{\lambda\sqrt{y^2+1}\sqrt{2\pi}} \exp\left(-(1/2) \left[\gamma + \delta \ln \left[y + \sqrt{y^2+1}\right]\right]^2\right) \quad \text{for } -\infty < x < \infty, \quad (47)$$

and distribution function

$$F(x) = \Phi\left\{\gamma + \delta \ln \left[y + \sqrt{y^2+1}\right]\right\}, \quad \text{for } -\infty < x < \infty, \quad (48)$$

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

$$F^{-1}(u) = \xi + \lambda(e^{t(u)} - e^{-t(u)})/2, \quad \text{for } 0 \leq u < 1, \quad (49)$$

where

$$t(u) = [\Phi^{-1}(u) - \gamma]/\delta. \quad (50)$$

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 double getGamma()
    Returns the value of  $\gamma$  for this object.

public double getDelta()
    Returns the value of  $\delta$  for this object.

public double getXi()
    Returns the value of  $\xi$  for this object.

public double getLambda()
    Returns the value of  $\lambda$  for this object.

public void setParams (double gamma, double delta, double xi, double lambda)
    Sets the value of the parameters  $\gamma$ ,  $\delta$ ,  $\xi$  and  $\lambda$  for this object.
```

LaplaceDist

Extends the class `ContinuousDistribution` for the *Laplace* distribution (see, e.g., [14, page 165]). It has location parameter θ and scale parameter $\phi > 0$. The density function is given by

$$f(x) = e^{-|x-\theta|/\phi}/(2\phi) \quad \text{for } -\infty < x < \infty. \quad (51)$$

The distribution function is

$$F(x) = \begin{cases} \frac{1}{2}e^{(x-\theta)/\phi} & \text{if } x \leq \theta, \\ 1 - \frac{1}{2}e^{(\theta-x)/\phi} & \text{otherwise,} \end{cases} \quad (52)$$

and its inverse is

$$F^{-1}(u) = \begin{cases} \phi \log(2u) + \theta & \text{if } 0 \leq u \leq \frac{1}{2}, \\ \theta - \phi \log(2(1-u)) & \text{otherwise.} \end{cases} \quad (53)$$

```
package umontreal.iro.lecuyer.probdist;
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  $\theta = \text{theta}$  and  $\phi = \text{phi}$ .
```

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 double getTheta()
    Returns the parameter  $\theta$ .

public double getPhi()
    Returns the parameter  $\phi$ .
```

LogisticDist

Extends the class `ContinuousDistribution` for the *logistic* distribution (e.g., [14, page 115]). It has location parameter α and scale parameter $\lambda > 0$. The density is

$$f(x) = \frac{\lambda e^{-\lambda(x-\alpha)}}{(1 + e^{-\lambda(x-\alpha)})^2} \quad \text{for } -\infty < x < \infty, \quad (54)$$

and the distribution function is

$$F(x) = \frac{1}{1 + e^{-\lambda(x-\alpha)}} \quad \text{for } -\infty < x < \infty. \quad (55)$$

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

$$F(x) = (1 + \tanh(x/2)) / 2. \quad (56)$$

The inverse distribution function is given by

$$F^{-1}(u) = \ln(u/(1-u))/\lambda + \alpha \quad \text{for } 0 \leq u < 1.$$

```
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 $\alpha = \text{alpha}$ and $\lambda = \text{lambda}$.

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 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.

LognormalDist

Extends the class `ContinuousDistribution` for the *lognormal* distribution [13]. It has scale parameter μ and shape parameter $\sigma > 0$. The density is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\ln(x)-\mu)^2/(2\sigma^2)} \quad \text{for } x > 0, \quad (57)$$

and 0 elsewhere. The distribution function is

$$F(x) = \Phi((\ln(x) - \mu)/\sigma) \quad \text{for } x > 0, \quad (58)$$

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

$$F^{-1}(u) = e^{\mu + \sigma\Phi^{-1}(u)} \quad \text{for } 0 \leq u < 1. \quad (59)$$

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 $\mu = \text{mu}$ and $\sigma = \text{sigma}$.

Methods

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

Computes the lognormal density function $f(x)$ in (57).

```
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 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., [13, page 80]). It has mean μ and variance σ^2 . Its density function is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)} \quad \text{for } -\infty < x < \infty, \quad (60)$$

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

$$F(x) = \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt \quad \text{for } -\infty < x < \infty. \quad (61)$$

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  $\mu = \text{mu}$  and  $\sigma = \text{sigma}$ .
```

Methods

```
public static double density (double mu, double sigma, double x)
    Computes the normal density function (60).

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  $\mu$  and variance  $\sigma^2$ . Uses the Chebyshev approximation proposed in [24], 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 [24].

```
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 [?]).

```
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  $\mu = \text{mu}$  and  $\sigma = \text{sigma}$ .
```

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  $\Phi$  is the standard normal distribution function,
    with mean 0 and variance 1. Uses Marsaglia et al's [20] 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  $\Phi$  is the standard normal distribution function,
    with mean 0 and variance 1. Uses Marsaglia et al's [20] 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  $\Phi$  is the standard normal distribution function,
    with mean 0 and variance 1. Uses the method of Marsaglia, Zaman, and Marsaglia [20],
    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 $\beta > 0$ [13, page 574]. The density for this type of Pareto distribution is

$$f(x) = \alpha\beta^\alpha x^{-(\alpha+1)} \quad \text{for } x \geq \beta, \quad (62)$$

and 0 otherwise. The distribution function is

$$F(x) = 1 - (\beta/x)^\alpha \quad \text{for } x \geq \beta, \quad (63)$$

and the inverse distribution function is

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

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

public class ParetoDist extends ContinuousDistribution
```

Constructors

```
public ParetoDist (double alpha)
    Constructs a ParetoDist object with parameters  $\alpha = \text{alpha}$  and  $\beta = 1$ .

public ParetoDist (double alpha, double beta)
    Constructs a ParetoDist object with parameters  $\alpha = \text{alpha}$  and  $\beta = \text{beta}$ .
```

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 double getAlpha()
    Returns the parameter  $\alpha$ .
```

```
public double getBeta()
```

Returns the parameter β .

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

Sets the parameter α and β for 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., [17, page 318]). The distribution function starts at $X_{(1)}$ and climbs linearly by $1/(n-1)$ between any two successive observations. The density is

$$f(x) = \frac{1}{(n-1)(X_{(i+1)} - X_{(i)})} \text{ for } X_{(i)} \leq x < X_{(i+1)} \text{ and } i = 1, 2, \dots, n-1. \quad (64)$$

The distribution function is

$$F(x) = \begin{cases} 0 & \text{for } x < X_{(1)}, \\ \frac{i-1}{n-1} + \frac{x - X_{(i)}}{(n-1)(X_{(i+1)} - X_{(i)})} & \text{for } X_{(i)} \leq x < X_{(i+1)} \text{ and } i < n, \\ 1 & \text{for } x \geq X_{(n)}, \end{cases} \quad (65)$$

whose inverse is

$$F^{-1}(u) = X_{(i)} + ((n-1)u - i + 1)(X_{(i+1)} - X_{(i)}) \quad (66)$$

for $(i-1)/(n-1) \leq u \leq i/(n-1)$ 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 (InputStreamReader in)
        throws IOException
```

Constructs a new empirical distribution using the observations read from the input stream 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 [14, page 362] with n degrees of freedom, where n is a positive integer. Its density is

$$f(x) = \frac{\Gamma((n+1)/2)}{\Gamma(n/2)\sqrt{\pi n}} \left(1 + \frac{x^2}{n}\right)^{-(n+1)/2} \quad \text{for } -\infty < x < \infty, \quad (67)$$

where $\Gamma(x)$ is the gamma function defined in (42).

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 (67) for a Student-t distribution with  $n$  degrees of freedom.

public static double cdf (int n, double x)
    Returns the approximation of [16, page 96] for the Student-t distribution function with  $n$  degrees of freedom. Gives at least 12 decimals of precision for  $n \leq 10^3$ , and at least 10 decimals for  $10^3 < n \leq 10^5$ .

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 [13])
```

$$2F(x) = \begin{cases} I_{n/2,1/2}(n/(n+x^2)) & \text{for } x < 0, \\ I_{1/2,n/2}(x^2/(n+x^2)) & \text{for } x \geq 0, \end{cases} \quad (68)$$

where $I_{\alpha,\beta}$ is the *beta* distribution function with parameters α and β (also called the incomplete *beta* ratio) defined in (21), 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 [11] and Figure L.28 of [3]).

```
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 [14, page 297] and [17, page 317]) with domain $[a, b]$ and *mode* (or shape parameter) m , where $a \leq m \leq b$. The density function is

$$f(x) = \begin{cases} \frac{2(x-a)}{(b-a)(m-a)} & \text{if } a \leq x \leq m, \\ \frac{2(b-x)}{(b-a)(b-m)} & \text{if } m \leq x \leq b, \\ 0 & \text{elsewhere,} \end{cases} \quad (69)$$

the distribution function is

$$F(x) = \begin{cases} 0 & \text{for } x < a, \\ \frac{(x-a)^2}{(b-a)(m-a)} & \text{if } a \leq x \leq m, \\ 1 - \frac{(b-x)^2}{(b-a)(b-m)} & \text{if } m \leq x \leq b, \\ 1 & \text{for } x > b, \end{cases} \quad (70)$$

and the inverse distribution function is given by

$$F^{-1}(u) = \begin{cases} a + \sqrt{(b-a)(m-a)u} & \text{if } 0 \leq u \leq \frac{m-a}{b-a}, \\ b - \sqrt{(b-a)(b-m)(1-u)} & \text{if } \frac{m-a}{b-a} \leq u \leq 1. \end{cases} \quad (71)$$

```
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 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 [14, page 276] over the interval $[a, b]$. Its density is

$$f(x) = 1/(b - a) \quad \text{for } a \leq x \leq b \quad (72)$$

and 0 elsewhere. The distribution function is

$$F(x) = (x - a)/(b - a) \quad \text{for } a \leq x \leq b \quad (73)$$

and its inverse is

$$F^{-1}(u) = a + (b - a)u \quad \text{for } 0 \leq u \leq 1. \quad (74)$$

```
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 (72).

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

Computes the uniform distribution function as in (73).

```
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 (74).

```
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 [13, page 628] with shape parameter $\alpha > 0$, location parameter δ , and scale parameter $\lambda > 0$. The density function is

$$f(x) = \alpha \lambda^\alpha (x - \delta)^{\alpha-1} e^{-(\lambda(x-\delta))^\alpha} \quad \text{for } x > \delta, \quad (75)$$

the distribution function is

$$F(x) = 1 - e^{-(\lambda(x-\delta))^\alpha} \quad \text{for } x > \delta, \quad (76)$$

and the inverse distribution function is

$$F^{-1}(u) = (-\ln(1-u))^{1/\alpha} / \lambda + \delta \quad \text{for } 0 \leq u < 1.$$

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

```
public class WeibullDist extends ContinuousDistribution
```

Constructors

```
public WeibullDist (double alpha)
```

Constructs a `WeibullDist` object with parameters $\alpha = \text{alpha}$, $\lambda = 1$, and $\delta = 0$.

```
public WeibullDist (double alpha, double lambda, double delta)
```

Constructs a `WeibullDist` object with parameters $\alpha = \text{alpha}$, $\lambda = \text{lambda}$, and $\delta = \text{delta}$.

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 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

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

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

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

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
```

```
    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.
```

```
    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] L. N. Bol’shev. Some applications of Pearson transformations. *Review of the Internat. Stat. Institute*, 32:14–16, 1964.
- [3] P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation*. Springer-Verlag, New York, second edition, 1987.
- [4] B. H. Camp. Approximation to the point binomial. *Ann. Math. Stat.*, 22:130–131, 1951.
- [5] T. G. Donnelly. Algorithm 462: Bivariate normal distribution. *Communications of the ACM*, 16(10):638, 1973.
- [6] W. Gautschi. Algorithm 222: Incomplete beta function ratios. *Communications of the ACM*, 7(3):143–144, 1964.
- [7] W. Gautschi. Certification of algorithm 222: Incomplete beta function ratios. *Communications of the ACM*, 7(3):244, 1964.
- [8] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer, New York, 1998.
- [9] A. Genz. Numerical computation of rectangular bivariate and trivariate normal and t probabilities. *Statistics and Computing*, 14:151–160, 2004. See <http://www.math.wsu.edu/faculty/genz/homepage>.
- [10] R. B. Goldstein. Algorithm 451: Chi-square quantiles. *Communications of the ACM*, 16:483–485, 1973.
- [11] G. W. Hill. Algorithm 395: Student’s t -distribution. *Communications of the ACM*, 13:617–619, 1970.
- [12] N. L. Johnson and S. Kotz. *Distributions in Statistics: Continuous Multivariate Distributions*. John Wiley, New York, 1972.
- [13] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume 1. Wiley, 2 edition, 1994.
- [14] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume 2. Wiley, 2 edition, 1995.
- [15] V. Kachitvichyanukul and B. Schmeiser. Computer generation of hypergeometric random variates. *J. Statist. Comput. Simul.*, 22:127–145, 1985.
- [16] W. J. Kennedy Jr. and J. E. Gentle. *Statistical Computing*. Dekker, New York, 1980.

- [17] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, third edition, 2000.
- [18] 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>.
- [19] K. V. Mardia and P. J. Zemroch. *Tables of the F and Related Distributions with Algorithms*. Academic Press, London, 1978.
- [20] 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.
- [21] W. Molenaar. *Approximations to the Poisson, Binomial and Hypergeometric Distribution Functions*, volume 31 of *Mathematical Center Tract*. Mathematisch Centrum, Amsterdam, 1970.
- [22] S. L. Moshier. Cephes math library, 2000. See <http://www.moshier.net>.
- [23] 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.
- [24] J. L. Schonfelder. Chebyshev expansions for the error and related functions. *Mathematics of Computation*, 32:1232–1240, 1978.