SSJ User's Guide

Package `probdistmulti`

Multivariate Probability Distributions

Version: March 8, 2007

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

Contents

Overview 2

General Classes **3**

DiscreteDistributionIntMulti 3

ContinuousDistributionMulti 4

ContinuousDistribution2Dim 5

Discrete Distributions over Integers **6**

MultinomialDist 6

NegativeMultinomialDist 8

2-Dimensional Continuous Distributions **10**

BiNormalDist 10

BiNormalGenzDist 13

BiNormalDonnellyDist 14

BiStudentDist 16

Continuous Distributions **18**

MultiNormalDist 18

DirichletDist 20

Overview

This package contains Java classes providing methods to compute mass, density, distribution and complementary distribution functions for some multi-dimensional discrete and continuous probability distributions. It does not generate random numbers for multivariate distributions; for that, see the package `randvarmulti`.

Distributions

We recall that the *distribution function* of a *continuous* random vector $X = \{x_1, x_2, \ldots, x_d\}$ with *density* $f(x_1, x_2, \ldots, x_d)$ over the d-dimensional space R^d is

$$
\begin{aligned}
F(x_1, x_2, \ldots, x_d) &= P[X_1 \le x_1, X_2 \le x_2, \ldots, X_d \le x_d] && (1)\\
&= \int_{-\infty}^{x_1} \int_{-\infty}^{x_2} \cdots \int_{-\infty}^{x_d} f(s_1, s_2, \ldots, s_d)\, ds_1 ds_2 \ldots ds_d && (2)
\end{aligned}
$$

while that of a *discrete* random vector X with *mass function* $\{p_1, p_2, \ldots, p_d\}$ over a fixed set of real numbers is

$$
\begin{aligned}
F(x_1, x_2, \ldots, x_d) &= P[X_1 \le x_1, X_2 \le x_2, \ldots, X_d \le x_d] && (3)\\
&= \sum_{i_1 \le x_1} \sum_{i_2 \le x_2} \cdots \sum_{i_d \le x_d} p(x_1, x_2, \ldots, x_d), && (4)
\end{aligned}
$$

where $p(x_1, x_2, \ldots, x_d) = P[X_1 = x_1, X_2 = x_2, \ldots, X_d = x_d]$. For a discrete distribution over the set of integers, one has

$$
\begin{aligned}
F(x_1, x_2, \ldots, x_d) &= P[X_1 \le x_1, X_2 \le x_2, \ldots, X_d \le x_d] && (5)\\
&= \sum_{s_1=-\infty}^{x_1} \sum_{s_2=-\infty}^{x_2} \cdots \sum_{s_d=-\infty}^{x_d} p(s_1, s_2, \ldots, s_d), && (6)
\end{aligned}
$$

where $p(s_1, s_2, \ldots, s_d) = P[X_1 = s_1, X_2 = s_2, \ldots, X_d = s_d]$.

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

$$
\bar{F}(x_1, x_2, \ldots, x_d) = P[X_1 \ge x_1, X_2 \ge x_2, \ldots, X_d \ge x_d]. \qquad (7)
$$

DiscreteDistributionIntMulti

Classes implementing multi-dimensional discrete distributions over the integers should inherit from this class. It specifies the signature of methods for computing the mass function (or probability) $p(x_1, x_2, \ldots, x_d) = P[X_1 = x_1, X_2 = x_2, \ldots, X_d = x_d]$ and the cumulative probabilities for a random vector X with a discrete distribution over the integers.

```
package umontreal.iro.lecuyer.probdistmulti;

public abstract class DiscreteDistributionIntMulti
```

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

Returns the probability mass function $p(x_1, x_2, \ldots, x_d)$, which should be a real number in $[0, 1]$.

```
public double cdf (int x[])
```

Computes the cumulative probability function F of the distribution evaluated at `x`, assuming the lowest values start at 0, i.e. computes

$$F(x_1, x_2, \ldots, x_d) = \sum_{s_1=0}^{x_1} \sum_{s_2=0}^{x_2} \cdots \sum_{s_d=0}^{x_d} p(s_1, s_2, \ldots, s_d).$$

Uses the naive implementation, is very inefficient and may underflows.

```
public int getDimension()
```

Returns the dimension d of the distribution.

```
public abstract double[] getMean();
```

Returns the mean vector of the distribution, defined as $\mu_i = E[X_i]$.

```
public abstract double[][] getCovariance();
```

Returns the variance-covariance matrix of the distribution, defined as
$\sigma_{ij} = E[(X_i - \mu_i)(X_j - \mu_j)]$.

```
public abstract double[][] getCorrelation();
```

Returns the correlation matrix of the distribution, defined as $\rho_{ij} = \sigma_{ij}/\sqrt{\sigma_{ii}\sigma_{jj}}$.

ContinuousDistributionMulti

Classes implementing continuous multi-dimensional distributions should inherit from this class. Such distributions are characterized by a *density* function $f(x_1, x_2, \ldots, x_d)$; thus the signature of a `density` method is supplied here. All array indices start at 0.

```
package umontreal.iro.lecuyer.probdistmulti;

public abstract class ContinuousDistributionMulti
```

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

Returns $f(x_1, x_2, \ldots, x_d)$, the probability density of X evaluated at the point x, where $x = \{x_1, x_2, \ldots, x_d\}$. The convention is that `x[i-1]` $= x_i$.

```
public int getDimension()
```

Returns the dimension d of the distribution.

```
public abstract double[] getMean();
```

Returns the mean vector of the distribution, defined as $\mu_i = E[X_i]$.

```
public abstract double[][] getCovariance();
```

Returns the variance-covariance matrix of the distribution, defined as
$\sigma_{ij} = E[(X_i - \mu_i)(X_j - \mu_j)]$.

```
public abstract double[][] getCorrelation();
```

Returns the correlation matrix of the distribution, defined as $\rho_{ij} = \sigma_{ij}/\sqrt{\sigma_{ii}\sigma_{jj}}$.

ContinuousDistribution2Dim

Classes implementing 2-dimensional 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 $\overline{F}(x, y)$, the upper `CDF`. The inverse function $F^{-1}(u)$ represents a curve $y = h(x)$ of constant u and it is not implemented.

```
package umontreal.iro.lecuyer.probdistmulti;

public abstract class ContinuousDistribution2Dim
                          extends ContinuousDistributionMulti
```

```
public int decPrec = 15;
```

Defines the target number of decimals of accuracy when approximating a distribution function, but there is *no guarantee* that this target is always attained.

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

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

```
public double density (double[] x)
```

Simply calls `density (x[0], x[1])`.

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

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

$$F(x, y) = P[X \le x, Y \le y] = \int_{-\infty}^{x} ds \int_{-\infty}^{y} dt\, f(s, t). \tag{8}$$

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

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

$$\overline{F}(x, y) = P[X \ge x, Y \ge y] = \int_{x}^{\infty} ds \int_{y}^{\infty} dt\, f(s, t). \tag{9}$$

```
public double cdf (double a1, double a2, double b1, double b2)
```

Computes the cumulative probability in the square region

$$P[a_1 \le X \le b_1,\ a_2 \le Y \le b_2] = \int_{a_1}^{b_1} dx \int_{a_2}^{b_2} dy\, f(x, y). \tag{10}$$

MultinomialDist

Implements the abstract class `DiscreteDistributionIntMulti` for the *multinomial* distribution with parameters n and $(p_1, \ldots, p_d)$. The probability mass function is [5]

$$P[X = (x_1, \ldots, x_d)] = \frac{n!}{\Pi_{i=1}^d x_i!} \Pi_{i=1}^d p_i^{x_i}, \tag{11}$$

where $\sum_{i=1}^d x_i = n$ and $\sum_{i=1}^d p_i = 1$.

```
package umontreal.iro.lecuyer.probdistmulti;

public class MultinomialDist extends DiscreteDistributionIntMulti
```

Constructors

`public MultinomialDist (int n, double p[])`

Creates a `MultinomialDist` object with parameters n and $(p_1, \ldots, p_d)$ such that $\sum_{i=1}^d p_i = 1$. We have $p_i =$ `p[i-1]`.

Methods

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

Computes the probability mass function (11) of the multinomial distribution with parameters n and $(p_1, \ldots, p_d)$ evaluated at x.

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

Computes the function F of the multinomial distribution with parameters n and $(p_1, \ldots, p_d)$ evaluated at x.

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

Computes the mean $E[X_i] = np_i$ of the multinomial distribution with parameters n and $(p_1, \ldots, p_d)$.

`public static double[][] getCovariance (int n, double[] p)`

Computes the covariance matrix of the multinomial distribution with parameters n and $(p_1, \ldots, p_d)$.

`public static double[][] getCorrelation (int n, double[] p)`

Computes the correlation matrix of the multinomial distribution with parameters n and $(p_1, \ldots, p_d)$.

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

Estimates and returns the parameters $[\hat{p_i}, \ldots, \hat{p_d}]$ of the multinomial distribution using the maximum likelihood method based on the m observations of d components in table $x[i][j]$, $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, d-1$.

The equations of the maximum likelihood are defined as

$$\hat{p_i} = \frac{\bar{X}_i}{N}.$$

```
public int getN()
```

Returns the parameter n of this object.

```
public double[] getP()
```

Returns the parameters $(p_1, \ldots, p_d)$ of this object.

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

Sets the parameters n and $(p_1, \ldots, p_d)$ of this object.

NegativeMultinomialDist

Implements the abstract class `DiscreteDistributionIntMulti` for the *negative multinomial* distribution with parameters $\gamma > 0$ and $(p_1,\ldots,p_d)$. The probability mass function is [5]

$$P[X = (x_1, \ldots, x_d)] = \frac{\Gamma\left(\gamma + \sum_{i=1}^{d} x_i\right) p_0^{\gamma} \Pi_{i=1}^{d} p_i^{x_i}}{\Gamma(\gamma) \Pi_{i=1}^{d} x_i!} \tag{12}$$

where $\sum_{i=1}^{d} p_i < 1$, $p_0 = 1 - \sum_{i=1}^{d} p_i$ and $p_i = \lambda_i / \left(1 + \sum_{i=1}^{d} \lambda_i\right)$.

```
package umontreal.iro.lecuyer.probdistmulti;
```

```
public class NegativeMultinomialDist extends DiscreteDistributionIntMulti
```

Constructors

```
public NegativeMultinomialDist (double gamma, double p[])
```

Creates a `NegativeMultinomialDist` object with parameters γ = `gamma` and $(p_1,\ldots,p_d)$ such that $\sum_{i=1}^{d} p_i < 1$, as described above. We have p_i = `p[i-1]`.

Methods

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

Computes the probability mass function (12) of the negative multinomial distribution with parameters γ and $(p_1,\ldots,p_d)$, evaluated at x.

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

Computes the cumulative probability function F of the negative multinomial distribution with parameters γ and $(p_1,\ldots,p_k)$, evaluated at x.

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

Computes the mean $E[X] = \gamma p_i / p_0$ of the negative multinomial distribution with parameters γ and $(p_1,\ldots,p_d)$.

```
public static double[][] getCovariance (double gamma, double p[])
```

Computes the covariance matrix of the negative multinomial distribution with parameters γ and $(p_1,\ldots,p_d)$.

```
public static double[][] getCorrelation (double gamma, double[] p)
```

Computes the correlation matrix of the negative multinomial distribution with parameters γ and $(p_1,\ldots,p_d)$.

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

Estimates the parameters $[\hat{\gamma},\ \hat{p_1},\ldots,\hat{p_d}]$ of the negative multinomial distribution using the maximum likelihood method based on the n observations of d components in table $x[i][j]$, $i = 0, 1, \ldots, n-1$ and $j = 0, 1, \ldots, d-1$.

The equations of the maximum likelihood are defined in [5]:

$$\begin{aligned}
\sum_{l=1}^{M} \frac{F_l}{(\hat{\gamma} + l - 1)} &= \ln\left(1 + \frac{1}{n\hat{\gamma}} \sum_{j=1}^{n} \Upsilon_j\right) \\
p_i &= \frac{\lambda_i}{1 + \sum_{j=1}^{d} \lambda_j} \qquad \text{for } i = 1, \ldots, d
\end{aligned}$$

where

$$\begin{aligned}
\lambda_i &= \frac{\sum_{j=1}^{n} X_{i,j}}{n\hat{\gamma}} \qquad \text{for } i = 1, \ldots, d \\
\Upsilon_j &= \sum_{i=1}^{d} X_{i,j} \qquad \text{for } j = 1, \ldots, n \\
F_l &= \frac{1}{n} \sum_{j=1}^{n} \mathbf{1}\{\Upsilon_j \geq l\} \qquad \text{for } l = 1, \ldots, M \\
M &= \max_j \{\Upsilon_j\}
\end{aligned}$$

```
public double getGamma()
```

Returns the parameter γ of this object.

```
public double[] getP()
```

Returns the parameters $(p_1,\ldots,p_d)$ of this object.

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

Sets the parameters γ and $(p_1,\ldots,p_d)$ of this object.

BiNormalDist

Extends the class `ContinuousDistribution2Dim` for the *bivariate normal* distribution [6, 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

$$f(x, y) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} e^{-T} \tag{13}$$

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

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

$$\Phi(\mu_1, \sigma_1, x, \mu_2, \sigma_2, y, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \int_{-\infty}^{x} dx \int_{-\infty}^{y} dy\, e^{-T}. \tag{14}$$

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

$$\overline{\Phi}(\mu_1, \sigma_1, x, \mu_2, \sigma_2, y, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \int_{x}^{\infty} dx \int_{y}^{\infty} dy\, e^{-T}. \tag{15}$$

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

$$\Phi(x, y, \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^{x} dx \int_{-\infty}^{y} dy\, e^{-S} \tag{16}$$

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

```
package umontreal.iro.lecuyer.probdistmulti;

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 $\rho =$ `rho`.

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

Constructs a `BiNormalDist` object with parameters $\mu_1 =$ `mu1`, $\mu_2 =$ `mu2`, $\sigma_1 =$ `sigma1`, σ_2 $=$ `sigma2` and $\rho =$ `rho`.

Methods

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

Computes the *standard binormal* density function (13) 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 (13) with parameters $\mu_1 =$ `mu1`, $\mu_2 =$ `mu2`, $\sigma_1 =$ `sigma1`, $\sigma_2 =$ `sigma2` and $\rho =$ `rho`.

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

Computes the standard *binormal* distribution (16) using the fast Drezner-Wesolowsky method described in [3]. The absolute error is expected to be smaller than $2 \cdot 10^{-7}$.

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

Computes the *binormal* distribution function (14) with parameters $\mu_1 =$ `mu1`, $\mu_2 =$ `mu2`, $\sigma_1 =$ `sigma1`, $\sigma_2 =$ `sigma2` and $\rho =$ `rho`. Uses the fast Drezner-Wesolowsky method described in [3]. The absolute error is expected to be smaller than $2 \cdot 10^{-7}$.

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

Computes the standard upper *binormal* distribution with $\mu_1 = \mu_2 = 0$ and $\sigma_1 = \sigma_2 = 1$. Uses the fast Drezner-Wesolowsky method described in [3]. The absolute error is expected to be smaller than $2 \cdot 10^{-7}$.

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

Computes the upper *binormal* distribution function (15) with parameters $\mu_1 =$ `mu1`, $\mu_2 =$ `mu2`, $\sigma_1 =$ `sigma1`, $\sigma_2 =$ `sigma2` and $\rho =$ `rho`. Uses the fast Drezner-Wesolowsky method described in [3]. The absolute error is expected to be smaller than $2 \cdot 10^{-7}$.

```
public static double[] getMean(double mu1, double sigma1,
                               double mu2, double sigma2, double rho)
```

Return the mean vector $E[X] = (\mu_1, \mu_2)$ of the binormal distribution.

```
public static double[][] getCovariance (double mu1, double sigma1,
                                        double mu2, double sigma2,
                                        double rho)
```

Return the covariance matrix of the binormal distribution.

```
public static double[][] getCorrelation (double mu1, double sigma1,
                                         double mu2, double sigma2,
                                         double rho)
```

Return the correlation matrix of the binormal distribution.

```
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 $\mu_1 =$ `mu1`, $\mu_2 =$ `mu2`, $\sigma_1 =$ `sigma1`, $\sigma_2 =$ `sigma2` and $\rho =$ `rho` of this object.

BiNormalGenzDist

Extends the class `BiNormalDist` for the *bivariate normal* distribution [6, page 84] using Genz's algorithm as described in [4].

```
package umontreal.iro.lecuyer.probdistmulti;
```

```
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 $\rho =$ `rho`.

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

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

Methods

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

Computes the standard *binormal* distribution (16) with the method described in [4]. 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 [6, page 84] using a translation of Donnelly's FORTRAN code in [2].

```
package umontreal.iro.lecuyer.probdistmulti;
```

```
public class BiNormalDonnellyDist extends BiNormalDist
```

Constructors

```
public BiNormalDonnellyDist (double rho, int ndig)
```

Constructor with default parameters $\mu_1 = \mu_2 = 0$, $\sigma_1 = \sigma_2 = 1$, correlation $\rho =$ `rho`, and $d =$ `ndig` digits of accuracy (the absolute error is smaller than 10^{-d}). Restriction: $d \le 15$.

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

Same as `BiNormalDonnellyDist (rho, 15)`.

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

Constructor with parameters $\mu_1 =$ `mu1`, $\mu_2 =$ `mu2`, $\sigma_1 =$ `sigma1`, $\sigma_2 =$ `sigma2`, $\rho =$ `rho`, and $d =$ `ndig` digits of accuracy. Restriction: $d \le 15$.

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

Same as `BiNormalDonnellyDist (mu1, sigma1, mu2, sigma2, rho, 15)`.

Methods

The following methods use the parameter `ndig` for the number of digits of absolute accuracy. If the same methods are called without the `ndig` parameter, a default value of `ndig` $= 15$ will be used.

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

Computes the standard *binormal* distribution (16) with the method described in [2], where `ndig` is the number of decimal digits of accuracy provided (`ndig` ≤ 15). The code was translated from the Fortran program written by T. G. Donnelly and copyrighted by the ACM (see `http://www.acm.org/pubs/copyright_policy/#Notice`). The absolute error is expected to be smaller than 10^{-d}, where $d =$ `ndig`.

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

Computes the *binormal* distribution function (14) with parameters $\mu_1 =$ `mu1`, $\mu_2 =$ `mu2`, σ_1 $=$ `sigma1`, $\sigma_2 =$ `sigma2`, correlation $\rho =$ `rho` and `ndig` decimal digits of accuracy.

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

Computes the upper *binormal* distribution function (15) with parameters $\mu_1 =$ `mu1`, $\mu_2 =$ `mu2`, $\sigma_1 =$ `sigma1`, $\sigma_2 =$ `sigma2`, $\rho =$ `rho` and `ndig` decimal digits of accuracy.

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

Computes the upper *standard binormal* distribution function (15) with parameters $\rho =$ `rho` and `ndig` decimal digits of accuracy.

BiStudentDist

Extends the class `ContinuousDistribution2Dim` for the *standard bivariate Student's t* distribution [6, page 132]. The correlation between X and Y is ρ and the number of degrees of freedom is ν. Its probability density is

$$f(x,y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \left[1 + \frac{x^2 - 2\rho xy + y^2}{\nu(1-\rho^2)}\right]^{-(\nu+2)/2}, \tag{17}$$

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

$$T_\nu(x,y,\rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^{x} dx \int_{-\infty}^{y} dy\, f(x,y). \tag{18}$$

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

$$\overline{T}_\nu(x,y,\rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{x}^{\infty} dx \int_{y}^{\infty} dy\, f(x,y). \tag{19}$$

```
package umontreal.iro.lecuyer.probdistmulti;
```

```
public class BiStudentDist extends ContinuousDistribution2Dim
```

Constructor

```
public BiStudentDist (int nu, double rho)
```

Constructs a `BiStudentDist` object with correlation $\rho =$ `rho` and $\nu =$ `nu` degrees of freedom.

Methods

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

Computes the standard bivariate Student's t density function (17) with correlation $\rho =$ `rho` and $\nu =$ `nu` degrees of freedom.

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

Computes the standard bivariate Student's t distribution (18) using the method described in [4]. 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 $\rho =$ `rho` and the number of degrees of freedom is $\nu =$ `nu`.

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

Computes the standard upper bivariate Student's t distribution (19).

`public static double[] getMean (int nu, double rho)`

Returns the mean vector $E[X] = (0, 0)$ of the bivariate Student's t distribution.

`public static double[][] getCovariance (int nu, double rho)`

Returns the covariance matrix of the bivariate Student's t distribution.

`public static double[][] getCorrelation (int nu, double rho)`

Returns the correlation matrix of the bivariate Student's t distribution.

`protected void setParams (int nu, double rho)`

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

MultiNormalDist

Implements the abstract class `ContinuousDistributionMulti` for the *multinormal* distribution with mean vector μ and covariance matrix Σ. The probability density is

$$P[X = (x_1, ..., x_d)] = \frac{1}{\sqrt{(2\pi)^d \det \Sigma}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right) \tag{20}$$

```
package umontreal.iro.lecuyer.probdistmulti;
```

```
public class MultiNormalDist extends ContinuousDistributionMulti
```

Constructors

```
public MultiNormalDist (double[] mu, double[][] sigma)
```

Methods

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

Computes the density (20) of the multinormal distribution with parameters $\mu =$ `mu` and $\Sigma =$ `sigma`, evaluated at `x`.

```
public int getDimension()
```

Returns the dimension d of the distribution.

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

Returns the mean $E[X] = \mu$ of the multinormal distribution with parameters μ and Σ.

```
public static double[][] getCovariance (double[] mu, double[][] sigma)
```

Computes the covariance matrix of the multinormal distribution with parameters μ and Σ.

```
public static double[][] getCorrelation (double[] mu, double[][] sigma)
```

Computes the correlation matrix of the multinormal distribution with parameters μ and Σ).

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

Estimates the parameters μ of the multinormal distribution using the maximum likelihood method based on the n observations of d components in table $x[i][j]$, $i = 0, 1, \ldots, n-1$ and $j = 0, 1, \ldots, d-1$.

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

Estimates the parameters Σ of the multinormal distribution using the maximum likelihood method based on the n observations of d components in table $x[i][j]$, $i = 0, 1, \ldots, n-1$ and $j = 0, 1, \ldots, d-1$.

`public double[] getMu()`

Returns the parameter μ of this object.

`public double getMu (int i)`

Returns the i-th component of 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.

DirichletDist

Implements the abstract class `ContinuousDistributionMulti` for the *Dirichlet* distribution with parameters (α_1,...,α_d), $\alpha_i > 0$. The probability density is

$$P[X = (x_1, \ldots, x_d)] = \frac{\Gamma(\alpha_0)\Pi_{i=1}^{d} x_i^{\alpha_i - 1}}{\Pi_{i=1}^{d}\Gamma(\alpha_i)} \tag{21}$$

where $x_i \geq 0$, $\sum_{i=1}^{d} x_i = 1$, $\alpha_0 = \sum_{i=1}^{d} \alpha_i$, and Γ is the Gamma function.

```
package umontreal.iro.lecuyer.probdistmulti;

public class DirichletDist extends ContinuousDistributionMulti
```

Constructors

```
public DirichletDist (double[] alpha)
```

Methods

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

Computes the density (21) of the Dirichlet distribution with parameters (α_1,...,α_d).

```
public static double[][] getCovariance (double[] alpha)
```

Computes the covariance matrix of the Dirichlet distribution with parameters (α_1,...,α_d).

```
public static double[][] getCorrelation (double[] alpha)
```

Computes the correlation matrix of the Dirichlet distribution with parameters (α_1,...,α_d).

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

Estimates and returns the parameters [$\hat{\alpha_1}, \ldots, \hat{\alpha_d}$] of the Dirichlet distribution using the maximum likelihood method based on the n observations of d components in table $x[i][j]$, $i = 0, 1, \ldots, n-1$ and $j = 0, 1, \ldots, d-1$.

The equations of the maximum likelihood are defined in [1, Technical appendix]

$$L(\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_k) \quad = \quad n\left(G(\alpha_0) - \sum_{i=1}^{k} G(\hat{\alpha}_i)\right) + \sum_{i=1}^{k}(\hat{\alpha}_i - 1)Z_i$$

where G is the logarithm of the gamma function and

$$\begin{aligned} \alpha_0 &= \sum_{i=1}^{k} \hat{\alpha}_i \\ Z_i &= \sum_{j=1}^{n} \ln(X_{i,j}) \qquad \text{for } i = 1, \ldots, k. \end{aligned}$$

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

Computes the mean $E[X] = \alpha_i/\alpha_0$ of the Dirichlet distribution with parameters $(\alpha_1,\ldots,\alpha_d)$, where $\alpha_0 = \sum_{i=1}^{d} \alpha_i$.

`public double[] getAlpha()`

Returns the parameters $(\alpha_1,\ldots,\alpha_d)$ of this object.

`public double getAlpha (int i)`

Returns the ith component of the alpha vector.

`public void setParams (double[] alpha)`

Sets the parameters $(\alpha_1,\ldots,\alpha_d)$ of this object.

References

[1] A. N. Avramidis, A. Deslauriers, and P. L'Ecuyer. Modeling daily arrivals to a telephone call center. *Management Science*, 50(7):896–908, 2004.

[2] T. G. Donnelly. Algorithm 462: Bivariate normal distribution. *Communications of the ACM*, 16(10):638, 1973.

[3] Z. Drezner and G. O. Wesolowsky. On the computation of the bivariate normal integral. *Journal of Statistical Computation and Simulation*, 35:101–107, 1990.

[4] 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`.

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

[6] N. L. Johnson and S. Kotz. *Distributions in Statistics: Continuous Multivariate Distributions*. John Wiley, New York, 1972.