SSJ User's Guide

Package `randvarmulti`

Generating Random Vectors

Version: February 25, 2009

This package is a multivariate version of the package `randvar`. It implements random number generators for some (nonuniform) multivariate distributions.

Contents

Overview 2
RandomMultivariateGen 3
MultinormalGen 4
MultinormalCholeskyGen 6
MultinormalPCAGen 8
MultiNormalGen 10
DirichletGen 12

Overview

This package provides a collection of classes for non-uniform random variate generation, very similar to `randvar`, but for multivariate distributions.

RandomMultivariateGen

This class is the multivariate counterpart of `RandomVariateGen`. It is the base class for general random variate generators over the d-dimensional real space $\mathbb{R}^d$. It specifies the signature of the `nextPoint` method, which is normally called to generate a random vector from a given distribution. Contrary to univariate distributions and generators, here the inversion method is not well defined, so we cannot construct a multivariate generator simply by passing a multivariate distribution and a stream; we must specify a generating method as well. For this reason, this class is abstract. Generators can be constructed only by invoking the constructor of a subclass. This is an important difference with `RandomVariateGen`.

```
package umontreal.iro.lecuyer.randvarmulti;

public abstract class RandomMultivariateGen
```

Methods

```
abstract public void nextPoint (double[] p);
```

Generates a random point p using the the stream contained in this object.

```
public void nextArrayOfPoints (double[][] v, int start, int n)
```

Generates n random points. These points are stored in the array `v`, starting at index `start`. Thus `v[start][i]` contains coordinate i of the first generated point. By default, this method calls `nextPoint` n times, but one can override it in subclasses for better efficiency. The array argument `v[][d]` must have d elements reserved for each generated point before calling this method.

```
public int getDimension()
```

Returns the dimension of this multivariate generator (the dimension of the random points).

```
public RandomStream getStream()
```

Returns the `RandomStream` used by this object.

```
public void setStream (RandomStream stream)
```

Sets the `RandomStream` used by this object to `stream`.

MultinormalGen

Extends `RandomMultivariateGen` for a *multivariate normal* (or *multinormal*) distribution [1]. The d-dimensional multivariate normal distribution with mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and (symmetric positive-definite) covariance matrix $\boldsymbol{\Sigma}$, denoted $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, has density

$$f(\mathbf{X}) = \frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma})}} \exp\left(-(\mathbf{X}-\boldsymbol{\mu})^{\mathsf{t}}\boldsymbol{\Sigma}^{-1}(\mathbf{X}-\boldsymbol{\mu})/2\right),$$

for all $\mathbf{X} \in \mathbb{R}^d$, and $\mathbf{X}^{\mathsf{t}}$ is the transpose vector of $\mathbf{X}$. If $\mathbf{Z} \sim N(\mathbf{0}, \mathbf{I})$ where $\mathbf{I}$ is the identity matrix, $\mathbf{Z}$ is said to have the *standard multinormal* distribution. For the special case $d = 2$, if the random vector $\mathbf{X} = (X_1, X_2)^{\mathsf{t}}$ has a bivariate normal distribution, then it has mean $\boldsymbol{\mu} = (\mu_1, \mu_2)^{\mathsf{t}}$, and covariance matrix

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

if and only if $\mathrm{Var}[X_1] = \sigma_1^2$, $\mathrm{Var}[X_2] = \sigma_2^2$, and the linear correlation between X_1 and X_2 is ρ, where $\sigma_1 > 0$, $\sigma_2 > 0$, and $-1 \le \rho \le 1$. $\boxed{1}$

```
package umontreal.iro.lecuyer.randvarmulti;

public class MultinormalGen extends RandomMultivariateGen
```

Constructors

```
public MultinormalGen (NormalGen gen1, int d)
```

Constructs a generator with the standard multinormal distribution (with $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma} = \mathbf{I}$) in d dimensions. Each vector $\mathbf{Z}$ will be generated via d successive calls to `gen1`, which must be a *standard normal* generator.

```
protected MultinormalGen (NormalGen gen1, double[] mu,
                          DoubleMatrix2D sigma)
```

Constructs a multinormal generator with mean vector `mu` and covariance matrix `sigma`. The mean vector must have the same length as the dimensions of the covariance matrix, which must be symmetric and positive-definite. If any of the above conditions is violated, an exception is thrown. The vector $\mathbf{Z}$ is generated by calling d times the generator `gen1`, which must be *standard normal*.

```
protected MultinormalGen (NormalGen gen1, double[] mu, double[][] sigma)
```

Equivalent to `MultinormalGen (gen1, mu, new DenseDoubleMatrix2D (sigma))`.

[1] From Richard: Rajouter un constructeur `MultinormalGen (RandomStream, int d)` ?

Methods

`public double[] getMu()`

Returns the mean vector used by this generator.

`public double getMu (int i)`

Returns the i-th component of the mean vector for this generator.

`public void setMu (double[] mu)`

Sets the mean vector to `mu`.

`public void setMu (int i, double mui)`

Sets the i-th component of the mean vector to `mui`.

`public DoubleMatrix2D getSigma()`

Returns the covariance matrix $\boldsymbol{\Sigma}$ used by this generator.

`public void nextPoint (double[] p)`

Generates a point from this multinormal distribution.

MultinormalCholeskyGen

Extends `MultinormalGen` for a *multivariate normal* distribution [1], generated via a Cholesky decomposition of the covariance matrix. The covariance matrix $\mathbf{\Sigma}$ is decomposed (by the constructor) as $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^{\mathsf{t}}$ where $\mathbf{A}$ is a lower-triangular matrix (this is the Cholesky decomposition), and $\mathbf{X}$ is generated via

$$\mathbf{X} = \boldsymbol{\mu} + \mathbf{A}\mathbf{Z},$$

where $\mathbf{Z}$ is a d-dimensional vector of independent standard normal random variates, and $\mathbf{A}^{\mathsf{t}}$ is the transpose of $\mathbf{A}$. The covariance matrix $\mathbf{\Sigma}$ must be positive-definite, otherwise the Cholesky decomposition will fail. The decomposition method uses the `CholeskyDecomposition` class in `colt`.

```
package umontreal.iro.lecuyer.randvarmulti;

   import cern.colt.matrix.DoubleMatrix2D;
   import cern.colt.matrix.impl.DenseDoubleMatrix2D;
   import cern.colt.matrix.linalg.CholeskyDecomposition;


public class MultinormalCholeskyGen extends MultinormalGen
```

Constructors

```
public MultinormalCholeskyGen (NormalGen gen1, double[] mu,
                               double[][] sigma)
```

Equivalent to `MultinormalCholeskyGen(gen1, mu, new DenseDoubleMatrix2D(sigma))`.

```
public MultinormalCholeskyGen (NormalGen gen1, double[] mu,
                               DoubleMatrix2D sigma)
```

Constructs a multinormal generator with mean vector `mu` and covariance matrix `sigma`. The mean vector must have the same length as the dimensions of the covariance matrix, which must be symmetric and positive-definite. If any of the above conditions is violated, an exception is thrown. The vector $\mathbf{Z}$ is generated by calling d times the generator `gen1`, which must be a *standard normal* 1-dimensional generator.

Methods

```
public DoubleMatrix2D getCholeskyDecompSigma()
```

Returns the lower-triangular matrix $\mathbf{A}$ in the Cholesky decomposition of $\mathbf{\Sigma}$.

```
public void setSigma (DoubleMatrix2D sigma)
```

Sets the covariance matrix $\mathbf{\Sigma}$ of this multinormal generator to `sigma` (and recomputes $\mathbf{A}$).

```
public static void nextPoint (NormalGen gen1, double[] mu,
                              double[][] sigma, double[] p)
```

Equivalent to `nextPoint(gen1, mu, new DenseDoubleMatrix2D(sigma), p)`.

```
public static void nextPoint (NormalGen gen1, double[] mu,
                              DoubleMatrix2D sigma, double[] p)
```

Generates a d-dimensional vector from the multinormal distribution with mean vector `mu` and covariance matrix `sigma`, using the one-dimensional normal generator `gen1` to generate the coordinates of $\mathbf{Z}$, and using the Cholesky decomposition of $\boldsymbol{\Sigma}$. The resulting vector is put into `p`. Note that this static method will be very slow for large dimensions, since it computes the Cholesky decomposition at every call. It is therefore recommended to use a `MultinormalCholeskyGen` object instead, if the method is to be called more than once.

```
public void nextPoint (double[] p)
```

Generates a point from this multinormal distribution. This is much faster than the static method as it computes the singular value decomposition matrix only once in the constructor.

MultinormalPCAGen

Extends `MultinormalGen` for a *multivariate normal* distribution [1], generated via the method of principal components analysis (PCA) of the covariance matrix. The covariance matrix $\mathbf{\Sigma}$ is decomposed (by the constructor) as $\mathbf{\Sigma} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathsf{t}}$ where $\mathbf{V}$ is an orthogonal matrix and $\mathbf{\Lambda}$ is the diagonal matrix made up of the eigenvalues of $\mathbf{\Sigma}$. $\mathbf{V}^{\mathsf{t}}$ is the transpose matrix of $\mathbf{V}$. The eigenvalues are ordered from the largest (λ_1) to the smallest (λ_d). The random multinormal vector $\mathbf{X}$ is generated via

$$\mathbf{X} = \boldsymbol{\mu} + \mathbf{A}\mathbf{Z},$$

where $\mathbf{A} = \mathbf{V}\sqrt{\mathbf{\Lambda}}$, and $\mathbf{Z}$ is a d-dimensional vector of independent standard normal random variates. The decomposition method uses the `SingularValueDecomposition` class in `colt`.

```
package umontreal.iro.lecuyer.randvarmulti;

import cern.colt.matrix.DoubleMatrix2D;
import cern.colt.matrix.linalg.SingularValueDecomposition;


public class MultinormalPCAGen extends MultinormalGen
```

Constructors

```
public MultinormalPCAGen (NormalGen gen1, double[] mu, double[][] sigma)
```

Equivalent to `MultinormalPCAGen(gen1, mu, new DenseDoubleMatrix2D(sigma))`.

```
public MultinormalPCAGen (NormalGen gen1, double[] mu,
                          DoubleMatrix2D sigma)
```

Constructs a multinormal generator with mean vector `mu` and covariance matrix `sigma`. The mean vector must have the same length as the dimensions of the covariance matrix, which must be symmetric and positive semi-definite. If any of the above conditions is violated, an exception is thrown. The vector $\mathbf{Z}$ is generated by calling d times the generator `gen1`, which must be a *standard normal* 1-dimensional generator.

Methods

```
public static DoubleMatrix2D decompPCA (double[][] sigma)
```

Computes the decomposition `sigma` $= \mathbf{\Sigma} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathsf{t}}$. Returns $\mathbf{A} = \mathbf{V}\sqrt{\mathbf{\Lambda}}$.

```
public static DoubleMatrix2D decompPCA (DoubleMatrix2D sigma)
```

Computes the decomposition `sigma` $= \mathbf{\Sigma} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathsf{t}}$. Returns $\mathbf{A} = \mathbf{V}\sqrt{\mathbf{\Lambda}}$.

```
public DoubleMatrix2D getPCADecompSigma()
```

Returns the matrix $\mathbf{A} = \mathbf{V}\sqrt{\mathbf{\Lambda}}$ of this object.

```
public static double[] getLambda (DoubleMatrix2D sigma)
```

Computes and returns the eigenvalues of `sigma` in decreasing order.

```
public double[] getLambda()
```

Returns the eigenvalues of $\mathbf{\Sigma}$ in decreasing order.

```
public void setSigma (DoubleMatrix2D sigma)
```

Sets the covariance matrix $\mathbf{\Sigma}$ of this multinormal generator to `sigma` (and recomputes $\mathbf{A}$).

```
public static void nextPoint (NormalGen gen1, double[] mu,
                              DoubleMatrix2D sigma, double[] p)
```

Generates a d-dimensional vector from the multinormal distribution with mean vector `mu` and covariance matrix `sigma`, using the one-dimensional normal generator `gen1` to generate the coordinates of $\mathbf{Z}$, and using the PCA decomposition of $\mathbf{\Sigma}$. The resulting vector is put into `p`. Note that this static method will be very slow for large dimensions, because it recomputes the singular value decomposition at every call. It is therefore recommended to use a `MultinormalPCAGen` object instead, if the method is to be called more than once.

```
public static void nextPoint (NormalGen gen1, double[] mu,
                              double[][] sigma, double[] p)
```

Equivalent to `nextPoint(gen1, mu, new DenseDoubleMatrix2D(sigma), p)`.

```
public void nextPoint (double[] p)
```

Generates a point from this multinormal distribution. This is much faster than the static method as it computes the singular value decomposition matrix only once in the constructor.

MultiNormalGen

THIS CLASS HAS BEEN RENAMED `MultinormalCholeskyGen` AND IS NOW DEPRECATED. IT WILL BE REMOVED IN FUTURE RELEASE.

```
package umontreal.iro.lecuyer.randvarmulti;

@Deprecated
public class MultiNormalGen extends RandomMultivariateGen
```

Constructors

`public MultiNormalGen (NormalGen gen1, int d)`

Constructs a standard d-dimensional multinormal generator, using the one-dimensional generator `gen1`.

`public MultiNormalGen (NormalGen gen1, double[] mu, DoubleMatrix2D sigma)`

Constructs a multinormal generator with mean vector `mu` and covariance matrix `sigma`, using the one-dimensional normal generator `gen1`. The mean vector must have the same length as the dimensions of the covariance matrix, which must be symmetric and positive-definite. If any of the above conditions is violated, an exception is thrown.

`public MultiNormalGen (NormalGen gen1, double[] mu, double[][] sigma)`

Equivalent to `MultiNormalGen (gen1, mu, new DenseDoubleMatrix2D (sigma))`.

Methods

`public double[] getMu()`

Returns the mean vector used by this generator.

`public double getMu (int i)`

Returns the i-th component of the mean vector for this generator.

`public void setMu (double[] mu)`

Sets the mean vector to `mu`.

`public void setMu (int i, double mui)`

Sets the i-th component of the mean vector to `mui`.

`public DoubleMatrix2D getSigma()`

Returns the covariance matrix $\mathbf{\Sigma}$ used by this generator.

`public DoubleMatrix2D getCholeskyDecompSigma()`

Returns the Cholesky decomposition of the covariance matrix used for generating the vectors. The returned matrix $\mathbf{A}$ is defined such that $\mathbf{\Sigma} = \mathbf{A}^{\mathbf{T}}\mathbf{A}$.

```
public void setSigma (DoubleMatrix2D sigma)
```

Sets the covariance matrix of this multinormal generator to `sigma`.

```
public static void nextPoint (NormalGen gen1, double[] mu,
                              DoubleMatrix2D sigma, double[] p)
```

Generates a point from the multinormal distribution with mean vector `mu`, and covariance matrix `sigma`, using the one-dimensional normal generator `gen1`. The resulting vector is put into `p`. Note that this static method may be slow for large dimension, because it needs to compute the Cholesky decomposition at every call. It is therefore recommended to use a `MultiNormalGen` object instead.

```
public static void nextPoint (NormalGen gen1, double[] mu,
                              double[][] sigma, double[] p)
```

Equivalent to `nextPoint (gen1, mu, new DenseDoubleMatrix2D (sigma), p)`.

```
public void nextPoint (double[] p)
```

Generates a point from the multinormal distribution.

DirichletGen

Extends `RandomMultivariateGen` for a *Dirichlet* [1] distribution. This distribution uses the parameters $\alpha_1, \ldots, \alpha_k$, and has density

$$f(x_1, \ldots, x_k) = \frac{\Gamma(\alpha_0) \prod_{i=1}^{k} x_i^{\alpha_i - 1}}{\prod_{i=1}^{k} \Gamma(\alpha_i)}$$

where $\alpha_0 = \sum_{i=1}^{k} \alpha_i$.

Here, the successive coordinates of the Dirichlet vector are generated $\boxed{2}$ via the class `GammaAcceptanceRejectionGen` in package `randvar`, using the same stream for all the uniforms.

```
package umontreal.iro.lecuyer.randvarmulti;

public class DirichletGen extends RandomMultivariateGen
```

Constructor

```
public DirichletGen (RandomStream stream, double[] alphas)
```

Constructs a new Dirichlet generator with parameters $\alpha_{i+1} =$ `alphas[i]`, for $i = 0, \ldots, k-1$, and the stream `stream`.

Methods

```
public double getAlpha (int i)
```

Returns the α_{i+1} parameter for this Dirichlet generator.

```
public static void nextPoint (RandomStream stream, double[] alphas,
                              double[] p)
```

Generates a new point from the Dirichlet distribution with parameters `alphas`, using the stream `stream`. The generated values are placed into `p`.

```
public void nextPoint (double[] p)
```

Generates a point from the Dirichlet distribution.

[2] From Pierre: How?

References

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