

Package: rangen (via r-universe)

September 10, 2026

Type Package

Title Random Number Generators and Utilities

Description Provides a collection of random number generators for common and custom distributions, along with utility functions for sampling and simulation.

Version 0.0.1

Date 2026-03-09

Maintainer Manos Papadakis <papadakm95@gmail.com>

License GPL-3

Imports Rcpp (>= 0.12.3)

LinkingTo Rcpp (>= 0.12.3), RcppArmadillo, zigg

Depends R (>= 3.5.0)

SystemRequirements C++17

NeedsCompilation yes

Repository <https://manospapadakis95.r-universe.dev>

Date/Publication 2026-09-10 11:13:00 UTC

RemoteUrl <https://github.com/manospapadakis95/rangen>

RemoteRef HEAD

RemoteSha 7cabdfe4a199aa06e24ab4d6345ab3d70a70d0c9

Contents

rangen-package	2
Angular central Gaussian random values simulation	2
Multivariate Laplace random values simulation	3
Multivariate normal and t random values simulation	5
Random integer values simulation	6
Random values simulation	8
rdiri	12
Simulation of random values from a Bingham distribution	13
Simulation of random values from a Bingham distribution with any symmetric matrix . .	14
Time functions	15

Index[17](#)

rangem-package	<i>Random Number Generators and Utilities</i>
----------------	---

Description

Provides a collection of random number generators for common and custom distributions, along with utility functions for sampling and simulation.

Details

Package:	rangem
Type:	Package
Version:	0.0.1
Date:	2026-03-09
License:	GPL-3

Maintainers

Manos Papadakis <papadakm95@gmail.com>

Author(s)

- Manos Papadakis <papadakm95@gmail.com>
- Michail Tsagris <mtsagris@uoc.gr>

Angular central Gaussian random values simulation
<i>Angular central Gaussian random values simulation</i>

Description

Angular central Gaussian random values simulation.

Usage

```
racg(n, sigma, seed = rangem::new.seed())
```

Arguments

<code>n</code>	The sample size, a numerical value.
<code>sigma</code>	The covariance matrix in R^d .
<code>seed</code>	If you want the same to be generated again use a seed for the generator, an integer number.

Details

The algorithm uses univariate normal random values and transforms them to multivariate via a spectral decomposition. The vectors are then scaled to have unit length.

Value

A matrix with the simulated data.

Author(s)

Michail Tsagris

R implementation and documentation: Michail Tsagris <mtsagris@uoc.gr>

References

Tyler D. E. (1987). Statistical analysis for the angular central Gaussian distribution on the sphere. *Biometrika* 74(3): 579-589.

See Also

[acg.mle](#), [rmvnorm](#), [rmvlaplace](#), [rmvt](#)

Examples

```
s <- cov( iris[, 1:4] )
#x <- racg(100, s)
#res<-acg.mle(x)
#res<-vmf.mle(x) ## the concentration parameter, kappa, is very low, close to zero, as expected.
```

Multivariate Laplace random values simulation

Multivariate Laplace random values simulation

Description

Multivariate Laplace random values simulation.

Usage

```
rmvlaplace(n, lam, mu, G, seed = rangen::new.seed())
```

Arguments

<code>n</code>	The sample size, a numerical value.
<code>lam</code>	The the parameter of the exponential distribution, a positive number.
<code>mu</code>	The mean vector.
<code>G</code>	A $d \times d$ covariance matrix with determinant 1.
<code>seed</code>	If you want the same to be generated again use a seed for the generator, an integer number.

Details

The algorithm uses univariate normal random values and transforms them to multivariate via a spectral decomposition.

Value

A matrix with the simulated data.

Author(s)

Michail Tsagris

R implementation and documentation: Michail Tsagris <mtsagris@uoc.gr>

References

Eltoft T., Kim T., and Lee T.W. (2006). On the multivariate laplace distribution. Signal Processing Letters, IEEE, 13(5):300-303.

See Also

[rmvnorm](#), [racg](#), [rmvt](#)

Examples

```
m <- colMeans(as.matrix(iris[, 1:4]))
s <- cov(iris[, 1:4])
s <- s / det(s)^0.25
lam <- 3
x <- rmvlaplace(100, lam, m, s)
```

Multivariate normal and t random values simulation

Multivariate normal and t random values simulation

Description

Multivariate normal and t random values simulation.

Usage

```
rmvnorm(n, mu, sigma, seed = rangen::new.seed())  
rmvt(n, mu, sigma, v, seed = rangen::new.seed())
```

Arguments

n	The sample size, a numerical value.
mu	The mean vector in R^d .
sigma	The covariance matrix in R^d .
v	The degrees of freedom.
seed	If you want the same to be generated again use a seed for the generator, an integer number.

Details

The algorithm uses univariate normal random values and transforms them to multivariate via a spectral decomposition. It is faster than the command "rmvnorm" available from MASS, and it allows for singular covariance matrices.

Value

A matrix with the simulated data.

Author(s)

Michail Tsagris

R implementation and documentation: Michail Tsagris <mtsagris@uoc.gr>

References

Aitchison J. (1986). The statistical analysis of compositional data. Chapman & Hall.

See Also

[racg](#), [rmvlaplace](#), [rmvt](#)

Examples

```
x <- as.matrix(iris[, 1:4])
m <- colMeans(x)
s <- cov(x)
y <- rmvnorm(1000, m, s)
res <- colMeans(y)
res <- cov(y)
y <- NULL
```

Random integer values simulation

Random integer values simulation

Description

Random integer values simulation.

Usage

```
Sample.int(n, size = n, replace = FALSE, seed = new.seed())
Sample(x, size = length(x), replace = FALSE, seed = new.seed())
colSample(x, size = rep_len(nrow(x), ncol(x)), replace = rep_len(FALSE, ncol(x)),
  parallel = FALSE, cores = 0, seed = new.seed())
rowSample(x, size = rep_len(ncol(x), nrow(x)), replace = rep_len(FALSE, nrow(x)),
  parallel = FALSE, cores = 0, seed = new.seed())
```

Arguments

<code>x</code>	A numeric vector for sampling or a matrix for column-row sampling.
<code>n</code>	This must be an integer value. The function will then draw random integer values from 1:n.
<code>size</code>	The number of integer values to sample.
<code>replace</code>	Do you want to sample with replacement? If yes, set this equal to TRUE.
<code>parallel</code>	Do you want to do it in parallel, for column - row major, in C++? TRUE or FALSE.
<code>cores</code>	Number of cores to use for parallelism. Valid only when argument parallel is set to TRUE. Default value is 0, which uses the OpenMP default number of threads..
<code>seed</code>	A seed object created by new.seed . To reproduce results, pass new.seed(x) with the same x. Not supported by the parallel versions.

Details

These functions provide flexible sampling utilities similar in purpose to R's base functions [sample.int](#) and [sample](#). Each function operates on a different structure:

- `Sample.int`: Generates a random sample of integers from 1 to `n`.
- `Sample`: Samples elements from the vector `x`.
- `colSample`: Performs column-wise sampling on a matrix or data frame, selecting `size[i]` elements from each column `i`.
- `rowSample`: Performs row-wise sampling on a matrix or data frame, selecting `size[i]` elements from each row `i`.

All functions support sampling with or without replacement. Parallel versions do not support seeding.

Value

`Sample.int`, `Sample`

A vector of sampled values.

`colSample`, `rowSample`

A matrix or data frame containing the sampled elements.

Author(s)

R implementation and documentation: Manos Papadakis <papadakm95@gmail.com>.

See Also

[sample](#), [sample.int](#)

Examples

```
# Sample integers from 1 to 10 with replacement (faster than base::sample.int)
x <- Sample.int(10, 1000, replace = TRUE)

# Sample from the vector 'x' (faster than base::sample)
xs <- Sample(x)

# Create a matrix and perform column-wise sampling
# 'size' must have the same length as the number of columns
mat <- matrix(1:20, nrow = 5, ncol = 4)
colSample(mat, size = rep(5, ncol(mat)), replace = rep(TRUE, ncol(mat)))

# Create a matrix and perform row-wise sampling
# 'size' must have the same length as the number of rows
rowSample(mat, size = rep(4, nrow(mat)), replace = rep(FALSE, nrow(mat)))
```

Random values simulation

Random values simulation from various distributions

Description

Functions to simulate random values from different probability distributions: uniform, beta, exponential, chi-squared, gamma, Cauchy, t-distribution, and geometric.

Usage

```

Runif(n, min = 0, max = 1, seed = new.seed())
Rbeta(n, alpha, beta, seed = new.seed())
Rexp(n, rate = 1, seed = new.seed())
Rchisq(n, df, seed = new.seed())
Rgamma(n, shape, rate = 1, seed = new.seed())
Rcauchy(n, location = 0, scale = 1, seed = new.seed())
Rt(n, df, ncp, seed = new.seed())
Rgeom(n, prob, seed = new.seed())
Rpareto(n, shape = 1, scale = 1, seed = new.seed())
Rfrechet(n, lambda = 1, mu = 0, sigma = 1, seed = new.seed())
Rlaplace(n, mu = 0, sigma = 1, seed = new.seed())
Rgumbel(n, mu = 0, sigma = 1, seed = new.seed())
Rarcsine(n, min = 0, max = 1, seed = new.seed())
Rnorm(n, mean = 0, sd = 1, seed = new.seed())
Runif.mat(nrow, ncol, min = 0, max = 1, seed = new.seed())
Rbeta.mat(nrow, ncol, alpha, beta, seed = new.seed())
Rexp.mat(nrow, ncol, rate = 1, seed = new.seed())
Rchisq.mat(nrow, ncol, df, seed = new.seed())
Rgamma.mat(nrow, ncol, shape, rate = 1, seed = new.seed())
Rcauchy.mat(nrow, ncol, location = 0, scale = 1, seed = new.seed())
Rt.mat(nrow, ncol, df, ncp, seed = new.seed())
Rgeom.mat(nrow, ncol, prob, seed = new.seed())
Rpareto.mat(nrow, ncol, shape = 1, scale = 1, seed = new.seed())
Rfrechet.mat(nrow, ncol, lambda = 1, mu = 0, sigma = 1, seed = new.seed())
Rlaplace.mat(nrow, ncol, mu = 0, sigma = 1, seed = new.seed())
Rgumbel.mat(nrow, ncol, mu = 0, sigma = 1, seed = new.seed())
Rarcsine.mat(nrow, ncol, min = 0, max = 1, seed = new.seed())
Rnorm.mat(nrow, ncol, mean = 0, sd = 1, seed = new.seed())
colRunif(nrow, ncol, min = rep_len(0, ncol), max = rep_len(1, ncol),
  seed = new.seed(), parallel = FALSE, cores = 0)
colRbeta(nrow, ncol, alpha, beta, seed = new.seed())
colRexp(nrow, ncol, rate = rep_len(1, ncol), seed = new.seed())
colRchisq(nrow, ncol, df, seed = new.seed())
colRgamma(nrow, ncol, shape, rate = rep_len(1, ncol), seed = new.seed())
colRgeom(nrow, ncol, prob, seed = new.seed())
colRcauchy(nrow, ncol, location = rep_len(0, ncol), scale = rep_len(1, ncol),

```

```

    seed = new.seed()
colRt(nrow, ncol, df, ncp, seed = new.seed())
colRpareto(nrow, ncol, shape = rep_len(1, ncol), scale = rep_len(1, ncol),
  seed = new.seed())
colRfrechet(nrow, ncol, lambda = rep_len(1, ncol), mu = rep_len(0, ncol),
  sigma = rep_len(1, ncol), seed = new.seed())
colRlaplace(nrow, ncol, mu = rep_len(0, ncol), sigma = rep_len(1, ncol),
  seed = new.seed())
colRgumbel(nrow, ncol, mu = rep_len(0, ncol), sigma = rep_len(1, ncol),
  seed = new.seed())
colRarcsine(nrow, ncol, min = rep_len(0, ncol), max = rep_len(1, ncol),
  seed = new.seed())
colRnorm(nrow, ncol, mean = rep_len(0, ncol), sd = rep_len(1, ncol),
  seed = new.seed())
new.seed(seed = nanoTime())

```

Arguments

n	The number of values to generate.
nrow	The number of rows.
ncol	The number of columns.
min	The lower value.
max	The upper value.
alpha	The shape parameter alpha.
beta	The shape parameter beta.
rate	Rgamma: The rate parameter.
df	Rt: The degrees of freedom.
lambda, shape	The shape parameter.
location, mu	The location parameter.
scale, sigma	The scale parameter.
ncp	The non-centrality parameter.
prob	The probability of success on each trial.
seed	For new.seed: a number used as the initial state of the generator. For all other functions: a seed object created by new.seed. To reproduce results, pass new.seed(x) with the same x.
parallel	colRunif only: should the columns be generated in parallel in C++? TRUE or FALSE.
cores	colRunif only: number of cores to use when parallel = TRUE. Default value is 0, which uses the OpenMP default number of threads.
mean	Vector of means.
sd	Vector of standard deviations.

Details

- `Runif`, `Runif.mat`, `colRunif`: generates random values from the uniform distribution, similar to R's built-in `runif` function. The type used is $\min + (\max - \min) \cdot U$, where U is a uniform random variable in the interval $(0, 1)$.
- `Rbeta`, `Rbeta.mat`, `colRbeta`: generates random values from the beta distribution with parameters α and β . The type used involves generating two gamma-distributed variables (X_1) and (X_2) , and returning $\frac{X_1}{X_1 + X_2}$, where (X_1) and (X_2) have the shape parameters α and β , respectively.
- `Rexp`, `Rexp.mat`, `colRexp`: generates random values from the exponential distribution with the specified rate parameter. The type used is $-\frac{\log(U)}{\text{rate}}$, where U is a uniform random variable in the interval $(0, 1)$.
- `Rchisq`, `Rchisq.mat`, `colRchisq`: generates random values from the chi-squared distribution with df degrees of freedom. The type used is the sum of the squares of df independent standard normal random variables, i.e., $Y_1^2 + Y_2^2 + \dots + Y_{df}^2$, where each (Y_i) is a standard normal random variable.
- `Rgamma`, `Rgamma.mat`, `colRgamma`: generates random values from the gamma distribution with shape and rate parameters. The type used for shape greater than or equal to 1 is the Marsaglia and Tsang method, which involves generating a variable (V) and returning $d \cdot V \cdot \text{rate}$, where (d) is a function of the shape and (V) is derived from a normal random variable. For shape less than 1, a combination of the uniform and exponential distributions is used, involving an acceptance-rejection method.
- `Rcauchy`, `Rcauchy.mat`, `colRcauchy`: generates random values from the Cauchy distribution with specified location and scale parameters. The type used for this is $\text{location} + \text{scale} \cdot \tan(\pi \cdot (U - 0.5))$, where U is a uniform random variable.
- `Rt`, `Rt.mat`, `colRt`: generates random values from the t-distribution with specified df degrees of freedom and an optional non-centrality parameter ncp . The type used is $\frac{Z}{\sqrt{\frac{Y}{df}}}$, where Z is a standard normal random variable and Y is a chi-squared random variable. If ncp is provided, the type used is $\frac{Z + ncp}{\sqrt{\frac{Y}{df}}}$.
- `Rgeom`, `Rgeom.mat`, `colRgeom`: generates random values from the geometric distribution with the specified probability $prob$. The type used is $\left\lfloor \frac{\log(U)}{\log(1 - prob)} \right\rfloor$, where U is a uniform random variable.
- `Rpareto`, `Rpareto.mat`, `colRpareto`: generates random values from the Pareto distribution with parameters $shape$ and $scale$. The type used is $\text{scale} \cdot (1 - U)^{-\frac{1}{\text{shape}}}$, where U is a uniform random variable in the interval $(0, 1)$.
- `Rfrechet`, `Rfrechet.mat`, `colRfrechet`: generates random values from the Frechet distribution with parameters $shape$, μ , and $scale$. The type used is $\mu + \text{scale} \cdot (-\log U)^{1/\text{shape}}$, where U is a uniform random variable in $(0, 1)$.
- `Rlaplace`, `Rlaplace.mat`, `colRlaplace`: generates random values from the Laplace distribution with location parameter μ and scale parameter σ . The type used is $\mu - \sigma \cdot \text{sign}(U) \cdot \log(1 - 2 \cdot |U|)$, where U is a uniform random variable in $(0, 1)$.
- `Rgumbel`, `Rgumbel.mat`, `colRgumbel`: generates random values from the Gumbel (type I extreme value) distribution with parameters μ and σ . The type used is $\mu - \sigma \cdot \log(-\log U)$, where U is a uniform random variable in $(0, 1)$.

- `Rarcsine`, `Rarcsine.mat`, `colRarcsine`: generates random values from the arcsine distribution over the interval $[\min, \max]$. The type used is $\min + (\max - \min) \cdot \sin^2(\pi U/2)$, where U is a uniform random variable in $(0, 1)$.
- `new.seed`: creates and returns a seed object of class "seed" that holds the generator state. Its fields `initialState` and `state` can be read and set with `$`. Seeding is not supported by the parallel versions of column and row sampling.

Value

Each function, for example `Runif`, returns a vector with simulated values from the respective distribution. Each function.mat, for example `Runif.mat`, returns a matrix with simulated values from the respective distribution. Each `colFunction`, for example `colRunif`, returns a matrix with column major simulated values from the respective distribution. `new.seed` returns an object of class "seed".

Author(s)

R implementation and documentation: Manos Papadakis <papadakm95@gmail.com>.

See Also

[runif](#), [rbeta](#), [rexp](#), [rchisq](#), [rgamma](#), [rcauchy](#), [rt](#), [rgeom](#)

Examples

```
# Scalar draws
x_unif    <- Runif(5, 0, 1)
x_beta    <- Rbeta(5, 2, 5)
x_exp     <- Rexp(5, 1.5)
x_chisq   <- Rchisq(5, 4)
x_gamma   <- Rgamma(5, 2, 2)
x_cauchy  <- Rcauchy(5, 0, 1)
x_t       <- Rt(5, df = 5, ncp = 2)
x_geom    <- Rgeom(5, 0.5)
x_pareto  <- Rpareto(5, shape = 2, scale = 1)
x_frechet <- Rfrechet(5, lambda = 1, mu = 0, sigma = 1)
x_laplace <- Rlaplace(5, mu = 0, sigma = 1)
x_gumbel  <- Rgumbel(5, mu = 0, sigma = 1)
x_arcsine <- Rarcsine(5, min = 0, max = 1)
x_norm    <- Rnorm(5)

#matrices
x_unif    <- Runif.mat(5,2, 0, 1)
x_beta    <- Rbeta.mat(5,2, 2, 5)
x_exp     <- Rexp.mat(5,2, 1.5)
x_chisq   <- Rchisq.mat(5,2, 4)
x_gamma   <- Rgamma.mat(5,2, 2, 2)
x_cauchy  <- Rcauchy.mat(5,2, 0, 1)
x_t       <- Rt.mat(5,2, df = 5, ncp = 2)
x_geom    <- Rgeom.mat(5,2, 0.5)
x_pareto  <- Rpareto.mat(5,2, shape = 2, scale = 1)
```

```

x_frechet <- Rfrechet.mat(5,2, lambda = 1, mu = 0, sigma = 1)
x_laplace <- Rlaplace.mat(5,2, mu = 0, sigma = 1)
x_gumbel <- Rgumbel.mat(5,2, mu = 0, sigma = 1)
x_arcsine <- Rarcsine.mat(5,2, min = 0, max = 1)
x_norm <- Rnorm.mat(5,2)

# Column-wise (vectorized by column) draws
x_col_unif <- colRunif(5, 2, min = c(0, 1), max = c(1, 2))
x_col_beta <- colRbeta(5, 2, alpha = c(2, 5), beta = c(5, 2))
x_col_exp <- colRexp(5, 2, rate = c(1.5, 2.0))
x_col_chisq <- colRchisq(5, 2, df = c(4, 5))
x_col_gamma <- colRgamma(5, 2, shape = c(2, 3), rate = c(2, 1))
x_col_cauchy <- colRcauchy(5, 2, location = c(0, 1), scale = c(1, 2))
x_col_t <- colRt(5, 2, df = c(5, 6), ncp = c(2, 1))
x_col_geom <- colRgeom(5, 2, prob = c(0.5, 0.3))
x_col_pareto <- colRpareto(5, 2, shape = c(2, 3), scale = c(1, 1))
x_col_frechet <- colRfrechet(5, 2, lambda = c(1, 2), mu = c(0, 0), sigma = c(1, 1))
x_col_laplace <- colRlaplace(5, 2, mu = c(0, 1), sigma = c(1, 2))
x_col_gumbel <- colRgumbel(5, 2, mu = c(0, 1), sigma = c(1, 2))
x_col_arcsine <- colRarcsine(5, 2, min = c(0, 0.5), max = c(1, 1.5))
x_col_norm <- colRnorm(5, 2)

```

rdiri

Dirichlet random values simulation

Description

Simulates random values from a Dirichlet distribution.

Usage

```
rdiri(n, a, b = rep(1, length(a)), seed = new.seed())
```

Arguments

n	The number of observations (rows) to generate.
a	A numeric vector of positive concentration parameters, one per component.
b	A numeric vector of positive rate parameters for the underlying gamma draws, one per component.
seed	A seed object created by new.seed .

Details

For each row, independent values $X_j \sim \text{Gamma}(a_j, b_j)$ are drawn and divided by their sum. With the default b, the result follows a Dirichlet(a) distribution. Unequal values in b give a scaled Dirichlet distribution, not a Dirichlet.

Value

A matrix with n rows and $\text{length}(a)$ columns. Each row sums to 1.

Author(s)

Manos Papadakis <papadakm95@gmail.com>.

See Also

[Rgamma](#), [colRgamma](#)

Examples

```
x <- rdiri(10, c(2, 3, 5))
rowSums(x)
```

Simulation of random values from a Bingham distribution
Simulating from a Bingham distribution

Description

Simulation from a Bingham distribution using the code suggested by Kent et al. (2013).

Usage

```
rbing(n, lam)
```

Arguments

<code>n</code>	Sample size.
<code>lam</code>	Eigenvalues of the diagonal symmetric matrix of the Bingham distribution. See details for more information on this.

Details

The user must have calculated the eigenvalues of the diagonal symmetric matrix of the Bingham distribution. The function accepts the $q-1$ eigenvalues only. This means, that the user must have subtracted the lowest eigenvalue from the rest and give the non zero ones. The function uses rejection sampling.

Value

A matrix with the simulated data.

Author(s)

Michail Tsagris

R implementation and documentation: Michail Tsagris <mtsagris@uoc.gr> and Manos Papadakis <papadakm95@gmail.com>

References

Kent J.T., Ganeiber A.M. and Mardia K.V. (2013). A new method to simulate the Bingham and related distributions in directional data analysis with applications. <http://arxiv.org/pdf/1310.8110v1.pdf>

C.J. Fallaize and T. Kypraios (2014). Exact Bayesian Inference for the Bingham Distribution. Statistics and Computing (No volum assigned yet). <http://arxiv.org/pdf/1401.2894v1.pdf>

See Also

[Rnorm](#)

Examples

```
x <- rbing( 100, c(1, 0.6, 0.1) )
x
```

Simulation of random values from a Bingham distribution with any symmetric matrix

Simulation of random values from a Bingham distribution with any symmetric matrix

Description

Simulation of random values from a Bingham distribution with any symmetric matrix.

Usage

```
rbingham(n, A)
```

Arguments

n	Sample size.
A	A symmetric matrix.

Details

The eigenvalues of the $q \times q$ symmetric matrix A are calculated and the smallest of them is subtracted from the rest. The $q - 1$ non zero eigenvalues are then passed to [rbing](#). The generated data are then right multiplied by V^T , where V is the matrix of eigenvectors of the matrix A .

Value

A matrix with the simulated data.

Author(s)

Michail Tsagris

R implementation and documentation: Michail Tsagris <mtsagris@uoc.gr> and Manos Papadakis <papadakm95@gmail.com>

References

Kent J.T., Ganeiber A.M. and Mardia K.V. (2013). A new method to simulate the Bingham and related distributions in directional data analysis with applications. <http://arxiv.org/pdf/1310.8110v1.pdf>
C.J. Fallaize and T. Kypraios (2014). Exact Bayesian Inference for the Bingham Distribution. Statistics and Computing (No volum assigned yet). <http://arxiv.org/pdf/1401.2894v1.pdf>

See Also

[Rnorm](#)

Examples

```
A <- cov( iris[, 1:4] )  
x <- rbingham(100, A)  
x
```

Time functions

Time functions

Description

Functions to get the current time in different units.

Usage

```
nanoTime()
```

Details

- nanoTime: The current nanoseconds of the system.

Value

Each function returns a numeric value.

Author(s)

Manos Papadakis <papadakm95@gmail.com>.

See Also

[runif](#), [rbeta](#), [rexp](#), [rchisq](#), [rgamma](#), [rcauchy](#), [rt](#), [rgeom](#)

Examples

```
x <- nanoTime()
```

Index

- * **Angular central Gaussian distribution**

- Angular central Gaussian random values simulation, [2](#)

- * **directional data**

- Angular central Gaussian random values simulation, [2](#)

- * **multivariate Laplace distribution**

- Multivariate Laplace random values simulation, [3](#)

- * **multivariate normal distribution**

- Multivariate normal and t random values simulation, [5](#)

- * **random values simulation**

- Angular central Gaussian random values simulation, [2](#)
 - Multivariate Laplace random values simulation, [3](#)
 - Multivariate normal and t random values simulation, [5](#)

acg.mle, [3](#)

Angular central Gaussian random values simulation, [2](#)

colRarcsine (Random values simulation), [8](#)

colRbeta (Random values simulation), [8](#)

colRcauchy (Random values simulation), [8](#)

colRchisq (Random values simulation), [8](#)

colRexp (Random values simulation), [8](#)

colRfrechet (Random values simulation), [8](#)

colRgamma, [13](#)

colRgamma (Random values simulation), [8](#)

colRgeom (Random values simulation), [8](#)

colRgumbel (Random values simulation), [8](#)

colRlaplace (Random values simulation), [8](#)

colRnorm (Random values simulation), [8](#)

colRpareto (Random values simulation), [8](#)

colRt (Random values simulation), [8](#)

colRunif (Random values simulation), [8](#)

colSample (Random integer values simulation), [6](#)

Multivariate Laplace random values simulation, [3](#)

Multivariate normal and t random values simulation, [5](#)

nanoTime (Time functions), [15](#)

new.seed, [6](#), [12](#)

new.seed (Random values simulation), [8](#)

racg, [4](#), [5](#)

racg (Angular central Gaussian random values simulation), [2](#)

Random integer values simulation, [6](#)

Random values simulation, [8](#)

rangem-package, [2](#)

Rarcsine (Random values simulation), [8](#)

Rbeta (Random values simulation), [8](#)

rbeta, [11](#), [16](#)

rbing, [14](#)

rbing (Simulation of random values from a Bingham distribution), [13](#)

rbingham (Simulation of random values from a Bingham distribution with any symmetric matrix), [14](#)

Rcauchy (Random values simulation), [8](#)

rcauchy, [11](#), [16](#)

Rchisq (Random values simulation), [8](#)

rchisq, [11](#), [16](#)

rdiri, [12](#)

Rexp (Random values simulation), [8](#)

rexp, [11](#), [16](#)

Rfrechet (Random values simulation), [8](#)

Rgamma, [13](#)

Rgamma (Random values simulation), [8](#)

rgamma, [11](#), [16](#)

Rgeom(Random values simulation), 8
rgeom, 11, 16
Rgumbel(Random values simulation), 8
Rlaplace(Random values simulation), 8
rmvlaplace, 3, 5
rmvlaplace(Multivariate Laplace
random values simulation), 3
rmvnorm, 3, 4
rmvnorm(Multivariate normal and t
random values simulation), 5
rmvt, 3–5
rmvt(Multivariate normal and t random
values simulation), 5
Rnorm, 14, 15
Rnorm(Random values simulation), 8
rowSample(Random integer values
simulation), 6
Rpareto(Random values simulation), 8
Rt(Random values simulation), 8
rt, 11, 16
Runif(Random values simulation), 8
runif, 10, 11, 16

Sample(Random integer values
simulation), 6
sample, 7
sample.int, 7
Simulation of random values from a
Bingham distribution, 13
Simulation of random values from a
Bingham distribution with any
symmetric matrix, 14

Time functions, 15