Package: rangen (via r-universe)

October 28, 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

Author Manos Papadakis [aut, cre, cph], Michail Tsagris [ctb], Omar Alzeley [ctb]

Repository https://cran.r-universe.dev

Date/Publication 2026-03-16 16:30:02 UTC

RemoteUrl https://github.com/cran/rangen

RemoteRef HEAD

RemoteSha e4407d1bb3af9aed4788890bab366664ba7da2b9

Contents

rangen-package 2
Random integer values simulation 2
Random values simulation 4
Time functions 8

Index **9**

rangen-package *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.

Details

Package:	rangen
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>

Random integer values simulation
Random integer values simulation

Description

Random integer values simulation.

Usage

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

Arguments

`x`	A numeric vector for sampling or a matrix for column-row sampling.
`n`	This must be an integer value. The function will then draw random integer values from 1:n.
`size`	The number of integer values to sample.
`replace`	Do you want to sample with replacement? If yes, set this equal to TRUE.
`parallel`	Do you want to do it in parallel, for column - row major, in C++? TRUE or FALSE.
`cores`	Number of cores to use for parallelism. Valid only when argument parallel is set to TRUE. Default value is 0 and it means the maximum supported cores.

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)
Rbeta(n, alpha, beta)
Rexp(n, rate = 1)
Rchisq(n, df)
Rgamma(n, shape, rate = 1)
Rcauchy(n, location = 0, scale = 1)
Rt(n, df, ncp)
Rgeom(n, prob)
Rpareto(n, shape = 1, scale = 1)
Rfrechet(n, lambda = 1, mu = 0, sigma = 1)
Rlaplace(n, mu = 0, sigma = 1)
Rgumble(n, mu = 0, sigma = 1)
Rarcsine(n, min = 0, max = 1)
Rnorm(n, mean = 0, sd = 1)
Runif.mat(nrow, ncol, min = 0, max = 1)
Rbeta.mat(nrow, ncol, alpha, beta)
Rexp.mat(nrow, ncol, rate = 1)
Rchisq.mat(nrow, ncol, df)
Rgamma.mat(nrow, ncol, shape, rate = 1)
Rcauchy.mat(nrow, ncol, location = 0, scale = 1)
Rt.mat(nrow, ncol, df, ncp)
```

```
Rgeom.mat(nrow, ncol, prob)
Rpareto.mat(nrow, ncol, shape = 1, scale = 1)
Rfrechet.mat(nrow, ncol, lambda = 1, mu = 0, sigma = 1)
Rlaplace.mat(nrow, ncol, mu = 0, sigma = 1)
Rgumble.mat(nrow, ncol, mu = 0, sigma = 1)
Rarcsine.mat(nrow, ncol, min = 0, max = 1)
Rnorm.mat(nrow, ncol, mean = 0, sd = 1)
colRunif(nrow, ncol, min = rep_len(0, ncol), max = rep_len(1, ncol))
colRbeta(nrow, ncol, alpha, beta)
colRexp(nrow, ncol, rate = rep_len(1, ncol))
colRchisq(nrow, ncol, df)
colRgamma(nrow, ncol, shape, rate = rep_len(1, ncol))
colRgeom(nrow, ncol, prob)
colRcauchy(nrow, ncol, location = rep_len(0, ncol), scale = rep_len(1, ncol))
colRt(nrow, ncol, df, ncp)
colRpareto(nrow, ncol, shape = rep_len(1, ncol), scale = rep_len(1, ncol))
colRfrechet(nrow, ncol, lambda = rep_len(1, ncol),
mu = rep_len(0, ncol), sigma = rep_len(1, ncol))
colRlaplace(nrow, ncol, mu = rep_len(0, ncol), sigma = rep_len(1, ncol))
colRgumble(nrow, ncol, mu = rep_len(0, ncol), sigma = rep_len(1, ncol))
colRarcsine(nrow, ncol, min = rep_len(0, ncol), max = rep_len(1, ncol))
colRnorm(nrow, ncol, mean = rep_len(0, ncol), sd = rep_len(1, ncol))
setSeed(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	A single value, interpreted as an integer.
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 `alpha` and `beta`. 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 `alpha` and `beta`, 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., $Y_1^2 + Y_2^2 + \cdots + Y_{df}^2$, 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 $d \cdot V \cdot$ 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 location + 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 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`, `mu`, and `scale`. The type used is μ + 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 `mu` and scale parameter `sigma`. 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).
- `Rgumble`, `Rgumble.mat`, `colRgumble`: generates random values from the Gumbel (type I extreme value) distribution with parameters `mu` and `sigma`. 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).
- `setSeed`: Set the seed for the Rngs. Not working with parallel version of column - row sample.

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.

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_gumblet  <- Rgumble(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_gumblet  <- Rgumble.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_gumble  <- colRgumble(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)
```

`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

colRarcsine *(Random values simulation)*, 4
colRbeta *(Random values simulation)*, 4
colRcauchy *(Random values simulation)*, 4
colRchisq *(Random values simulation)*, 4
colRexp *(Random values simulation)*, 4
colRfrechet *(Random values simulation)*, 4
colRgamma *(Random values simulation)*, 4
colRgeom *(Random values simulation)*, 4
colRgumble *(Random values simulation)*, 4
colRlaplace *(Random values simulation)*, 4
colRnorm *(Random values simulation)*, 4
colRpareto *(Random values simulation)*, 4
colRt *(Random values simulation)*, 4
colRunif *(Random values simulation)*, 4
colSample *(Random integer values simulation)*, 2

nanoTime *(Time functions)*, 8

Random integer values simulation, 2
Random values simulation, 4
rangen-package, 2
Rarcsine *(Random values simulation)*, 4
Rbeta *(Random values simulation)*, 4
rbeta, *7, 8*
Rcauchy *(Random values simulation)*, 4
rcauchy, *7, 8*
Rchisq *(Random values simulation)*, 4
rchisq, *7, 8*
Rexp *(Random values simulation)*, 4
rexp, *7, 8*
Rfrechet *(Random values simulation)*, 4
Rgamma *(Random values simulation)*, 4
rgamma, *7, 8*
Rgeom *(Random values simulation)*, 4
rgeom, *7, 8*
Rgumble *(Random values simulation)*, 4
Rlaplace *(Random values simulation)*, 4
Rnorm *(Random values simulation)*, 4
rowSample *(Random integer values simulation)*, 2
Rpareto *(Random values simulation)*, 4
Rt *(Random values simulation)*, 4
rt, *7, 8*
Runif *(Random values simulation)*, 4
runif, *6–8*

Sample *(Random integer values simulation)*, 2
sample, *3*
sample.int, *3*
setSeed *(Random values simulation)*, 4

Time functions, 8