

# Package ‘dqrng’

August 30, 2023

**Type** Package

**Title** Fast Pseudo Random Number Generators

**Version** 0.3.1

**Description** Several fast random number generators are provided as C++ header only libraries: The PCG family by O’Neill (2014 <<https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf>>) as well as Xoroshiro128+ and Xoshiro256+ by Blackman and Vigna (2018 <[arXiv:1805.01407](https://arxiv.org/abs/1805.01407)>). In addition fast functions for generating random numbers according to a uniform, normal and exponential distribution are included. The latter two use the Ziggurat algorithm originally proposed by Marsaglia and Tsang (2000, <[doi:10.18637/jss.v005.i08](https://doi.org/10.18637/jss.v005.i08)>). The fast sampling methods support unweighted sampling both with and without replacement. These functions are exported to R and as a C++ interface and are enabled for use with the default 64 bit generator from the PCG family, Xoroshiro128+ and Xoshiro256+ as well as the 64 bit version of the 20 rounds Threefry engine (Salmon et al., 2011, <[doi:10.1145/2063384.2063405](https://doi.org/10.1145/2063384.2063405)>) as provided by the package ‘sitmo’.

**License** AGPL-3 | file LICENSE

**Depends** R (>= 3.5.0)

**Imports** Rcpp (>= 0.12.16)

**LinkingTo** Rcpp, BH (>= 1.64.0-1), sitmo (>= 2.0.0)

**RoxygenNote** 7.2.3

**Suggests** testthat, knitr, rmarkdown, mvtnorm (>= 1.2-3), bench, sitmo

**VignetteBuilder** knitr

**URL** <https://daqana.github.io/dqrng/>, <https://github.com/daqana/dqrng>

**BugReports** <https://github.com/daqana/dqrng/issues>

**Encoding** UTF-8

**NeedsCompilation** yes

**Author** Ralf Stubner [aut, cre],  
daqana GmbH [cph],  
David Blackman [ctb],

Melissa O'Neill [ctb],  
 Sebastiano Vigna [ctb],  
 Aaron Lun [ctb],  
 Kyle Butts [ctb]

**Maintainer** Ralf Stubner <ralf.stubner@gmail.com>

**Repository** CRAN

**Date/Publication** 2023-08-30 19:40:02 UTC

## R topics documented:

dqrng-package . . . . .	2
dqrmvnorm . . . . .	3
dqRNGkind . . . . .	4
dqsample . . . . .	6
generateSeedVectors . . . . .	6
<b>Index</b>	<b>8</b>

---

dqrng-package	<i>dqrng: Fast Pseudo Random Number Generators</i>
---------------	--

---

## Description

Several fast random number generators are provided as C++ header only libraries: The PCG family by O'Neill (2014 <https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf>) as well as Xoroshiro128+ and Xoshiro256+ by Blackman and Vigna (2018 [arXiv:1805.01407](https://arxiv.org/abs/1805.01407)). In addition fast functions for generating random numbers according to a uniform, normal and exponential distribution are included. The latter two use the Ziggurat algorithm originally proposed by Marsaglia and Tsang (2000, [doi:10.18637/jss.v005.i08](https://doi.org/10.18637/jss.v005.i08)). The fast sampling methods support unweighted sampling both with and without replacement. These functions are exported to R and as a C++ interface and are enabled for use with the default 64 bit generator from the PCG family, Xoroshiro128+ and Xoshiro256+ as well as the 64 bit version of the 20 rounds Threefry engine (Salmon et al., 2011, [doi:10.1145/2063384.2063405](https://doi.org/10.1145/2063384.2063405)) as provided by the package 'sitmo'.

## Author(s)

**Maintainer:** Ralf Stubner <ralf.stubner@gmail.com>

Other contributors:

- daqana GmbH [copyright holder]
- David Blackman [contributor]
- Melissa O'Neill <oneill@pcg-random.org> [contributor]
- Sebastiano Vigna <vigna@acm.org> [contributor]
- Aaron Lun [contributor]
- Kyle Butts <kyle.butts@colorado.edu> [contributor]

**See Also**

Useful links:

- <https://daqana.github.io/dqrng/>
- <https://github.com/daqana/dqrng>
- Report bugs at <https://github.com/daqana/dqrng/issues>

---

dqrnorm

*Multivariate Distributions*

---

**Description**

Multivariate Distributions

**Usage**

```
dqrnorm(n, ...)
```

**Arguments**

n	number of observations
...	forwarded to <a href="#">rmvnorm</a> or <a href="#">rmvt</a>

**Value**

numeric matrix of multivariate normal distributed variables

**See Also**

[rmvnorm](#)

**Examples**

```
sigma <- matrix(c(4,2,2,3), ncol=2)
x <- dqrnorm(n=500, mean=c(1,2), sigma=sigma)
colMeans(x)
var(x)
plot(x)
```

---

`dqRNGkind`*R interface*

---

## Description

The dqrng package provides several fast random number generators together with fast functions for generating random numbers according to a uniform, normal and exponential distribution. These functions are modeled after the base functions `set.seed`, `RNGkind`, `runif`, `rnorm`, and `rexp`.

`dqrrademacher` uses a fast algorithm to generate random Rademacher variables (-1 and 1 with equal probability). To do so, it generates a random 64 bit integer and then uses each bit to generate a 0/1 variable. This generates 64 integers per random number generation.

## Usage

```
dqRNGkind(kind, normal_kind = "ignored")
```

```
dqrunif(n, min = 0, max = 1)
```

```
dqnorm(n, mean = 0, sd = 1)
```

```
dqrexp(n, rate = 1)
```

```
dqrrademacher(n)
```

```
dqset.seed(seed, stream = NULL)
```

## Arguments

<code>kind</code>	string specifying the RNG (see details)
<code>normal_kind</code>	ignored; included for compatibility with <code>RNGkind</code>
<code>n</code>	number of observations
<code>min</code>	lower limit of the uniform distribution
<code>max</code>	upper limit of the uniform distribution
<code>mean</code>	mean value of the normal distribution
<code>sd</code>	standard deviation of the normal distribution
<code>rate</code>	rate of the exponential distribution
<code>seed</code>	integer scalar to seed the random number generator, or an integer vector of length 2 representing a 64-bit seed. Maybe NULL, see details.
<code>stream</code>	integer used for selecting the RNG stream; either a scalar or a vector of length 2

## Details

Supported RNG kinds:

**pcg64** The default 64 bit variant from the PCG family developed by Melissa O'Neill. See <https://www.pcg-random.org/> for more details.

**Xoroshiro128+ and Xoshiro256+** RNGs developed by David Blackman and Sebastiano Vigna. They are used as default RNGs in Erlang and Lua. See <https://xoshiro.di.unimi.it/> for more details.

**Threefry** The 64 bit version of the 20 rounds Threefry engine as provided by [sitmo-package](#)

Xoroshiro128+ is the default since it is the fastest generator provided by this package.

The functions `dqrnorm` and `dqrexp` use the Ziggurat algorithm as provided by `boost.random`.

See [generateSeedVectors](#) for rapid generation of integer-vector seeds that provide 64 bits of entropy. These allow full exploration of the state space of the 64-bit RNGs provided in this package.

If the provided seed is NULL, a seed is generated from R's RNG without state alteration.

## Value

`dqrunif`, `dqrnorm`, and `dqrexp` return a numeric vector of length `n`. `dqrrademacher` returns an integer vector of length `n`.

## See Also

[set.seed](#), [RNGkind](#), [runif](#), [rnorm](#), and [rexp](#)

## Examples

```
library(dqrng)

# Set custom RNG.
dqRNGkind("Xoshiro256+")

# Use an integer scalar to set a seed.
dqset.seed(42)

# Use integer scalars to set a seed and the stream.
dqset.seed(42, 123)

# Use an integer vector to set a seed.
dqset.seed(c(31311L, 24123423L))

# Use an integer vector to set a seed and a scalar to select the stream.
dqset.seed(c(31311L, 24123423L), 123)

# Random sampling from distributions.
dqrunif(5, min = 2, max = 10)
dqrexp(5, rate = 4)
dqrnorm(5, mean = 5, sd = 3)
```

---

`dqsample`*Unbiased Random Samples and Permutations*

---

**Description**

Unbiased Random Samples and Permutations

**Usage**`dqsample(x, size, replace = FALSE, prob = NULL)``dqsample.int(n, size = n, replace = FALSE, prob = NULL)`**Arguments**

`x` either a vector of one or more elements from which to choose, or a positive integer.

`size` a non-negative integer giving the number of items to choose.

`replace` should sampling be with replacement?

`prob` a vector of probability weights for obtaining the elements of the vector being sampled.

`n` a positive number, the number of items to choose from.

**See Also**vignette("sample", package = "dqrng"), [sample](#) and [sample.int](#)

---

`generateSeedVectors`*Generate seed as a integer vector*

---

**Description**

Generate seed as a integer vector

**Usage**`generateSeedVectors(nseeds, nwords = 2L)`**Arguments**

`nseeds` Integer scalar, number of seeds to generate.

`nwords` Integer scalar, number of words to generate per seed.

**Details**

Each seed is encoded as an integer vector with the most significant bits at the start of the vector. Each integer vector is converted into an unsigned integer (in C++ or otherwise) by the following procedure:

1. Start with a sum of zero.
2. Add the first value of the vector.
3. Left-shift the sum by 32.
4. Add the next value of the vector, and repeat.

The aim is to facilitate R-level generation of seeds with sufficient randomness to cover the entire state space of pseudo-random number generators that require more than the ~32 bits available in an `int`. It also preserves the integer nature of the seed, thus avoiding problems with casting double-precision numbers to integers.

It is possible for the seed vector to contain `NA_integer_` values. This should not be cause for alarm, as R uses `-INT_MAX` to encode missing values in integer vectors.

**Value**

A list of length `n`, where each element is an integer vector that contains `nwords` words (i.e., `32*nwords` bits) of randomness.

**Author(s)**

Aaron Lun

**Examples**

```
generateSeedVectors(10, 2)
```

```
generateSeedVectors(5, 4)
```

# Index

dqrexp (dqRNGkind), 4  
dqrmvnorm, 3  
dqrng (dqrng-package), 2  
dqrng-package, 2  
dqRNGkind, 4  
dqrnorm (dqRNGkind), 4  
dqrrademacher (dqRNGkind), 4  
dqrunif (dqRNGkind), 4  
dqsample, 6  
dqset.seed (dqRNGkind), 4

generateSeedVectors, 5, 6

rexp, 4, 5  
rmvnorm, 3  
rmvt, 3  
RNGkind, 4, 5  
rnorm, 4, 5  
runif, 4, 5

sample, 6  
sample.int, 6  
set.seed, 4, 5