

# Package ‘qrng’

May 9, 2026

**Version** 0.0-11

**Encoding** UTF-8

**Title** (Randomized) Quasi-Random Number Generators

**Description** Functionality for generating (randomized) quasi-random numbers in high dimensions.

**Author** Marius Hofert [aut, cre],  
Christiane Lemieux [aut]

**Maintainer** Marius Hofert <mhofert@hku.hk>

**Depends** R (>= 3.0.0)

**Imports** utils

**Suggests** spacefillr, randtoolbox, copula, simsalapar

**License** GPL-2 | GPL-3

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2026-01-22 06:10:43 UTC

## Contents

qrng . . . . .	1
test_functions . . . . .	4
to_array . . . . .	6

<b>Index</b>	<b>8</b>
--------------	----------

---

qrng *Compute Quasi-Random Sequences*

---

## Description

Computing Korobov, generalize Halton and Sobol’ quasi-random sequences.

**Usage**

```

korobov(n, d = 1, generator, randomize = c("none", "shift"))
ghalton(n, d = 1, method = c("generalized", "halton"))
sobol (n, d = 1, randomize = c("none", "digital.shift", "Owen", "Faure.Tezuka",
                              "Owen.Faure.Tezuka"), seed, skip = 0, ...)

```

**Arguments**

n	number $n$ of points to be generated $\geq 2$ .
d	dimension $d$ .
generator	<b>numeric</b> of length $d$ or length 1 (in which case it is appropriately extended to length $d$ ). All numbers must be in $\{1, \dots, n\}$ and must be (coercible to) integers.
randomize	<b>character</b> string indicating the type of randomization for the point set. korobov() one of the following: <b>"none"</b> no randomization. <b>"shift"</b> a uniform random variate modulo 1. sobol() one of the following: "none" no randomization. <b>"digital.shift"</b> a digital shift. <b>"Owen"</b> calls <code>generate_sobol_owen_set()</code> from package <b>spacefillr</b> . <b>"Faure.Tezuka", "Owen.Faure.Tezuka"</b> calls <code>sobol()</code> from package <b>randtoolbox</b> with scrambling being 2 and 3, respectively. If randomize is a <b>logical</b> , then it is interpreted as "none" if FALSE and "digital.shift" if TRUE.
method	<b>character</b> string indicating which sequence is generated, generalized Halton or (plain) Halton.
seed	if provided, an integer used within <code>set.seed()</code> for the non-scrambling randomize methods (so "none" or "digital.shift") or passed to the underlying <code>generate_sobol_owen_set()</code> from package <b>spacefillr</b> (for "Owen") and <code>sobol()</code> from package <b>randtoolbox</b> for the scrambling methods. If not provided, the non-scrambling methods respect a global <code>set.seed()</code> but the scrambling methods do not (they then use a randomly generated one based on time and process identifier).
skip	number of initial points in the sequence to be skipped ( <code>skip = 0</code> means the sequence starts with at the origin). Note that for the scrambling methods this simply computes $n + \text{skip}$ points and omits the first <code>skip</code> -many.
...	additional arguments passed to <code>sobol()</code> from package <b>randtoolbox</b> for randomization methods "Faure.Tezuka" and "Owen.Faure.Tezuka".

**Details**

For `sobol()` examples see `demo(sobol_examples)`.

Note that these procedures call fast C code. The following restrictions apply:

**korobov()**  $n, d$  must be  $\leq 2^{31} - 1$ .

**ghalton()**  $n$  must be  $\leq 2^{32} - 1$  and  $d$  must be  $\leq 360$ .

`sobol()` if `randomize = "none"` or `randomize = "digital.shift"`,  $n$  must be  $\leq 2^{31} - 1$  and  $d$  must be  $\leq 16510$ .

The choice of parameters for `korobov()` is crucial for the quality of this quasi-random sequence (only basic sanity checks are conducted). For more details, see l'Ecuyer and Lemieux (2000).

The generalized Halton sequence uses the scrambling factors of Faure and Lemieux (2009).

### Value

`korobov()` and `ghalton()` return an  $(n, d)$ -matrix; for  $d = 1$  an  $n$ -vector is returned.

### Author(s)

Marius Hofert and Christiane Lemieux

### References

Faure, H., Lemieux, C. (2009). Generalized Halton Sequences in 2008: A Comparative Study. *ACM-TOMACS* **19**(4), Article 15.

l'Ecuyer, P., Lemieux, C. (2000). Variance Reduction via Lattice Rules. *Stochastic Models and Simulation*, 1214–1235.

Lemieux, C., Cieslak, M., Luttmer, K. (2004). RandQMC User's guide. See <https://www.math.uwaterloo.ca/~clemieux/randqmc/>

### Examples

```
n <- 1021 # prime
d <- 4 # dimension

## Korobov's sequence
generator <- 76 # see l'Ecuyer and Lemieux
u <- korobov(n, d = d, generator = generator)
pairs(u, gap = 0, pch = ".", labels = as.expression(
  sapply(1:d, function(j) bquote(italic(u[.(j)]))))))

## Randomized Korobov's sequence
set.seed(271)
u <- korobov(n, d = d, generator = generator, randomize = "shift")
pairs(u, gap = 0, pch = ".", labels = as.expression(
  sapply(1:d, function(j) bquote(italic(u[.(j)]))))))

## Generalized Halton sequence (randomized by definition)
set.seed(271)
u <- ghalton(n, d)
pairs(u, gap = 0, pch = ".", labels = as.expression(
  sapply(1:d, function(j) bquote(italic(u[.(j)]))))))

## For sobol() examples, see demo(sobol_examples)
```

test\_functions

*Test Functions***Description**

Functions for testing low-discrepancy sequences.

**Usage**

```
sum_of_squares(u)
sobol_g(u, copula = copula::indepCopula(dim = ncol(u)), alpha = 1:ncol(u), ...)
exceedance(x, q, p = 0.99, method = c("indicator", "individual.given.sum.exceeds",
                                     "sum.given.sum.exceeds"))
```

**Arguments**

u	( $n, d$ )-matrix containing $n$ $d$ -dimensional realizations (of a potential quasi-random number generator). For <code>sum_of_squares()</code> these need to be marginally standard uniform and for <code>sobol_g()</code> they need to follow the copula specified by <code>copula</code> .
copula	<a href="#">Copula</a> object for which the inverse Rosenblatt transformation exists.
alpha	vector of parameters of Sobol's $g$ test function.
...	additional arguments passed to the underlying <code>cCopula()</code> .
x	( $n, d$ )-matrix containing $n$ $d$ -dimensional realizations.
q	<b>"indicator"</b> $d$ -vector containing the componentwise thresholds; if a number it is recycled to a $d$ -vector. <b>"individual.given.sum.exceeds"</b> , <b>"sum.given.sum.exceeds"</b> threshold for the sum (row sums of $x$ ).
p	If $q$ is not provided, the probability $p$ is used to determine $q$ . <b>"indicator"</b> $d$ -vector containing the probabilities determining componentwise thresholds via empirical quantiles; if a number, it is recycled to a $d$ -vector. <b>"individual.given.sum.exceeds"</b> , <b>"sum.given.sum.exceeds"</b> probability determining the threshold for the sum (row sums of $x$ ) via the corresponding empirical quantile.
method	<a href="#">character</a> string indicating the type of exceedance computed (see Section Value below).

**Details**

For examples see the demo `man_test_functions`.

See `ES_np(<matrix>)` from **qrmtools** for another test function.

**Value**

sum\_of\_squares() returns an  $n$ -vector (`numeric(n)`) with the rowwise computed scaled sum of squares (theoretically integrating to 1).

sobol\_g() returns an  $n$ -vector (`numeric(n)`) with the rowwise computed Sobol' g functions.

exceedance()'s return value depends on method:

**"indicator"** returns indicators whether, componentwise,  $x$  exceeds the threshold determined by  $q$ .

**"individual.given.sum.exceeds"** returns all rows of  $x$  whose sum exceeds the threshold determined by  $q$ .

**"sum.given.sum.exceeds"** returns the row sums of those rows of  $x$  whose sum exceeds the threshold determined by  $q$ .

**Author(s)**

Marius Hofert and Christiane Lemieux

**References**

Radovic, I., Sobol', I. M. and Tichy, R. F. (1996). Quasi-Monte Carlo methods for numerical integration: Comparison of different low discrepancy sequences. *Monte Carlo Methods and Applications* **2**(1), 1–14.

Faure, H., Lemieux, C. (2009). Generalized Halton Sequences in 2008: A Comparative Study. *ACM-TOMACS* **19**(4), Article 15.

Owen, A. B. (2003). The dimension distribution and quadrature test functions. *Stat. Sinica* **13**, 1–17.

Sobol', I. M. and Asotsky, D. I. (2003). One more experiment on estimating high-dimensional integrals by quasi-Monte Carlo methods. *Math. Comput. Simul.* **62**, 255–263.

**Examples**

```
## Generate some (here: copula, pseudo-random) data
library(copula)
set.seed(271)
cop <- claytonCopula(iTau(claytonCopula(), tau = 0.5)) # Clayton copula
U <- rCopula(1000, copula = cop)

## Compute sum of squares test function
mean(sum_of_squares(U)) # estimate of  $E(3(\sum_{j=1}^d U_j^2)/d)$ 

## Compute the Sobol' g test function
if(packageVersion("copula") >= "0.999-20")
  mean(sobol_g(U)) # estimate of  $E(\langle \text{Sobol's g function} \rangle)$ 

## Compute an exceedance probability
X <- qnorm(U)
mean(exceedance(X, q = qnorm(0.99))) # fixed threshold q
mean(exceedance(X, p = 0.99)) # empirically estimated marginal p-quantiles as thresholds
```

```
## Compute 99% expected shortfall for the sum
mean(exceedance(X, p = 0.99, method = "sum.given.sum.exceeds"))
## Or use ES_np(X, level = 0.99) from 'qrmtools'
```

---

to\_array

*Compute Matrices to Arrays*


---

### Description

Converting higher-dimensional matrices of quasi-random numbers to arrays of specific formats.

### Usage

```
to_array(x, f, format = c("(n*f,d)", "(n,f,d)"))
```

### Arguments

`x`  $(n,fd)$ -matrix of quasi-random numbers to be converted.  
`f` factor  $f \geq 1$  dividing `ncol{x}`.  
`format` **character** string indicating the output format to which `x` should be converted.

### Details

`to_array()` is helpful for converting quasi-random numbers to time series paths.

### Value

$(n * f, d)$ -**matrix** or  $(n, f, d)$ -**array** depending on the chosen format.

### Author(s)

Marius Hofert

### See Also

[korobov\(\)](#), [ghalton\(\)](#), [sobol\(\)](#).

### Examples

```
## Basic call
N <- 4 # replications
n <- 3 # time steps
d <- 2 # dimension
set.seed(271) # note: respected for the choice of 'randomize'
x <- sobol(N, d = n * d, randomize = "digital.shift") # higher-dim. Sobol'
stopifnot(dim(to_array(x, f = n)) == c(N * n, d)) # conversion and check
stopifnot(dim(to_array(x, f = n, format = "(n,f,d)")) == c(N, n, d))

## See how the conversion is done
```

```
(x <- matrix(1:(N * n * d), nrow = N, byrow = TRUE))  
to_array(x, f = n) # => (n * d)-column x was blocked in n groups of size d each
```

# Index

\* **distribution**

qrng, 1

\* **utilities**

test\_functions, 4

to\_array, 6

array, 6

cCopula, 4

character, 2, 4, 6

Copula, 4

exceedance (test\_functions), 4

generate\_sobol\_owen\_set, 2

ghalton, 6

ghalton (qrng), 1

korobov, 6

korobov (qrng), 1

logical, 2

matrix, 3, 6

numeric, 2, 5

qrng, 1

set.seed, 2

sobol, 2, 6

sobol (qrng), 1

sobol\_g (test\_functions), 4

sum\_of\_squares (test\_functions), 4

test\_functions, 4

to\_array, 6