# Package 'qrng'

June 21, 2018

**Version** 0.0-5

**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 <marius.hofert@uwaterloo.ca>

**Depends** R (>= 3.0.0)

**Imports**

**Suggests**

**Enhances**

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

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2018-06-21 17:37:54 UTC

## R topics documented:

qrng *Compute Quasi-Random Sequences*

## Description

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

## Usage

```
korobov(n, d = 1, generator, randomize = FALSE)
ghalton(n, d = 1, method = c("generalized", "halton"))
sobol  (n, d = 1, randomize = FALSE, skip = 0)
```

## Arguments

| | |
|---|---|
| n | Number $n$ of points to be generated $\geq 2$. |
| d | Dimension $d$. |
| generator | A numeric of length $d$ or length 1 (in which case it is appropriately extended to length $d$). All numbers must be in $\{1, \ldots, n\}$ and must be (coercible to) integers. |
| randomize | A logical indicating whether the point set should be randomized (for sobol(): a digital shift). |
| method | A character string indicating which sequence is generated, generalized Halton or (plain) Halton. |
| skip | number of initial terms in the sequence that are skipped (skip = 0 means the sequence starts with the origin). |

## Details

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()** 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).

See the example below on being careful about using skip > 0 when randomize = TRUE; in this case, choosing a wrong seed (or no seed) might lead to a bad sequence.

## 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/randq

## 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 = TRUE)
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)])))))

## Randomized Sobol sequence (with digital shift)
set.seed(271)
u <- sobol(n, d, randomize = TRUE)
pairs(u, gap = 0, pch = ".", labels = as.expression(
      sapply(1:d, function(j) bquote(italic(u[.(j)])))))

## Check whether a Sobol' sequence of size 2*n equals one of size n
## and, concatenated, one of size n with the first n numbers skipped
f <- function(n)
{
    set.seed(271)
    a <- sobol(2*n, randomize = TRUE)
    set.seed(271)
    b1 <- sobol(n, randomize = TRUE)
    set.seed(271)
    b2 <- sobol(n, randomize = TRUE, skip = n)
    all(all.equal(apply(cbind(a, c(b1, b2)), 1, diff), rep(0, 2*n)))
```

```
}
stopifnot(sapply(1:10, f)) # check for n = 1, ..., 10

## Careful when using skip > 0 and randomize = TRUE => seed matters!

## Drawing all points at once (works, of course)
n <- 32
set.seed(5)
U.2n <- sobol(2*n, d = 2, randomize = TRUE)
plot(U.2n, main = "All points generated at once",
     xlab = expression(U[1]), ylab = expression(U[2]))

## Drawing successively with the same seed (typically the right approach)
set.seed(5)
U.n.1 <- sobol(n, d = 2, randomize = TRUE)
set.seed(5) # => same seed
U.n.2 <- sobol(n, d = 2, randomize = TRUE, skip = n)
U.2n.same.seed <- rbind(U.n.1, U.n.2)
plot(U.2n.same.seed,
     main = "Drawing successively, with the same seed",
     xlab = expression(U[1]), ylab = expression(U[2]))
stopifnot(all.equal(U.2n, U.2n.same.seed)) # sanity check

## Drawing successively but with different seeds (typically the wrong approach)
set.seed(5)
U.n.1 <- sobol(n, d = 2, randomize = TRUE, skip = 0)
set.seed(22)
U.n.2 <- sobol(n, d = 2, randomize = TRUE, skip = n)
U.2n.different.seed <- rbind(U.n.1, U.n.2)
plot(U.2n.different.seed,
     main = "Drawing successively, with different seeds",
     xlab = expression(U[1]), ylab = expression(U[2]))
```

# Index