Package ‘spacefillr’
February 23, 2021

Type Package
Title Space-Filling Random and Quasi-Random Sequences
Version 0.2.0
Maintainer Tyler Morgan-Wall <tylermw@gmail.com>


License MIT + file LICENSE
Imports Rcpp (>= 1.0.0)
LinkingTo Rcpp
Encoding UTF-8
LazyData true
RoxygenNote 7.1.1

URL https://github.com/tylermorganwall/spacefillr
BugReports https://github.com/tylermorganwall/spacefillr/issues
SystemRequirements C++11

NeedsCompilation yes
Author Tyler Morgan-Wall [aut, cph, cre]
  (<https://orcid.org/0000-0002-3131-3814>),
  Andrew Helmer [ctb, cph],
  Leonhard Grünschloß [ctb, cph],
  Eric Heitz [ctb, cph]

Repository CRAN
Date/Publication 2021-02-23 09:40:02 UTC
**R topics documented:**

- `generate_halton_faure_set` .................................................. 2
- `generate_halton_faure_single` ............................................ 3
- `generate_halton_random_set` ............................................... 3
- `generate_halton_random_single` ......................................... 4
- `generate_pj_set` ............................................................... 5
- `generate_pmj02bn_set` ....................................................... 6
- `generate_pmj02_set` ........................................................... 7
- `generate_pmjbn_set` ............................................................ 8
- `generate_pmj_set` ............................................................... 9
- `generate_sobol_owen_set` .................................................... 10
- `generate_sobol_set` ........................................................... 11

**Index**

---

**generate_halton_faure_set**

*Generate Halton Set (Faure Initialized)*

**Description**

Generate a set of values from a Faure Halton set.

**Usage**

```r
generate_halton_faure_set(n, dim)
```

**Arguments**

- `n` : The number of values (per dimension) to extract.
- `dim` : The number of dimensions of the sequence.

**Value**

An `n` x `dim` matrix listing all the

**Examples**

```r
#Generate a 2D sample:
points2d = generate_halton_random_set(n=1000, dim=2)
plot(points2d)

#Extract a separate pair of dimensions
points2d = generate_halton_random_set(n=1000, dim=10)
plot(points2d[,5:6])

#Integrate the value of pi by counting the number of randomly generated points that fall
#within the unit circle.
```
generate_halton_faure_single

Generate Halton Value (Faure Initialized)

Description
Generate a single value from a seeded Halton set, initialized with a Faure sequence.
Note: This is much slower than generating the entire set ahead of time.

Usage

\texttt{generate_halton_faure_single(i, dim)}

Arguments

- \texttt{i} \hspace{0.2cm} The element of the sequence to extract.
- \texttt{dim} \hspace{0.2cm} The dimension of the sequence to extract.

Value
A single numeric value representing the ‘i’th element in the ‘dim’ dimension.

Examples

\begin{verbatim}
#Generate a 3D sample:
point3d = c(generate_halton_faure_single(10, dim = 1),
generate_halton_faure_single(10, dim = 2),
generate_halton_faure_single(10, dim = 3))
point3d
\end{verbatim}

generate_halton_random_set

Generate Halton Set (Randomly Initialized)

Description
Generate a set of values from a seeded Halton set.

Usage

\texttt{generate_halton_random_set(n, dim, seed = 0)}
Arguments

- **n**: The number of values (per dimension) to extract.
- **dim**: The number of dimensions of the sequence.
- **seed**: Default ‘0’. The random seed.

Value

An ‘n’ x ‘dim’ matrix listing all the

Examples

```r
#Generate a 2D sample:
points2d = generate_halton_random_set(n=1000, dim=2)
plot(points2d)

#Change the seed and extract a separate pair of dimensions
points2d = generate_halton_random_set(n=1000, dim=10, seed=2)
plot(points2d[,5:6])

#Integrate the value of pi by counting the number of randomly generated points that fall
#within the unit circle.
pointset = matrix(generate_halton_random_set(10000, dim=2), ncol=2)

pi_estimate = 4*sum(pointset[,1] * pointset[,1] + pointset[,2] * pointset[,2] < 1)/10000
pi_estimate
```

---

**generate_halton_random_single**  
*Generate Halton Value (Randomly Initialized)*

Description

Generate a single value from a seeded Halton set.  
Note: This is much slower than generating the entire set ahead of time.

Usage

```r
generate_halton_random_single(i, dim, seed = 0)
```

Arguments

- **i**: The element of the sequence to extract.
- **dim**: The dimension of the sequence to extract.
- **seed**: Default ‘0’. The random seed.

Value

A single numeric value representing the ‘i’th element in the ‘dim’ dimension.
generate_pj_set

Examples

# Generate a 3D sample:
point3d = c(generate_halton_random_single(10, dim = 1),
            generate_halton_random_single(10, dim = 2),
            generate_halton_random_single(10, dim = 3))
point3d

# Change the random seed:
# 'Generate a 3D sample
point3d_2 = c(generate_halton_random_single(10, dim = 1, seed = 10),
              generate_halton_random_single(10, dim = 2, seed = 10),
              generate_halton_random_single(10, dim = 3, seed = 10))
point3d_2

---

generate_pj_set  Generate 2D Progressive Jittered Set

Description

Generate a set of values from a Progressive Jittered set.

Usage

generate_pj_set(n, seed = 0)

Arguments

n  The number of 2D values to extract.
seed  Default ‘0’. The random seed.

Value

An ‘n’ x ‘2’ matrix with all the calculated values from the set.

Examples

# Generate a 2D sample:
points2d = generate_pj_set(n=1000)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

# Generate a longer sequence of values from that set
points2d = generate_pj_set(n=1500)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

# Generate a new set by changing the seed
points2d = generate_pj_set(n=1500, seed=10)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

# 'Integrate the value of pi by counting the number of randomly generated points that fall
generate_pmj02bn_set

Generate 2D Progressive Multi-Jittered (0, 2) (with blue noise) Set

Description
Generate a set of values from a Progressive Multi-Jittered (0, 2) (with blue noise) set.

Usage
generate_pmj02bn_set(n, seed = 0)

Arguments
n The number of 2D values to extract.
seed Default ‘0’: The random seed.

Value
An ‘n’ x ‘2’ matrix with all the calculated values from the set.

Examples
#Generate a 2D sample:
points2d = generate_pmj02bn_set(n=1000)
plot(points2d, xlim=c(0,1),ylim=c(0,1))

#Generate a longer sequence of values from that set
points2d = generate_pmj02bn_set(n=1500)
plot(points2d, xlim=c(0,1),ylim=c(0,1))

#Generate a new set by changing the seed
points2d = generate_pmj02bn_set(n=1500,seed=10)
plot(points2d, xlim=c(0,1),ylim=c(0,1))

#Integrate the value of pi by counting the number of randomly generated points that fall
#within the unit circle.
pointset = generate_pmj02bn_set(10000)
pi_estimate = 4*sum(pointset[,1] * pointset[,1] + pointset[,2] * pointset[,2] < 1)/10000
pi_estimate
generate_pmj02_set

Generate 2D Progressive Multi-Jittered (0, 2) Set

Description

Generate a set of values from a Progressive Multi-Jittered (0, 2) set.

Usage

generate_pmj02_set(n, seed = 0)

Arguments

n 
The number of 2D values to extract.

seed 
Default ‘0’. The random seed.

Value

An ‘n’ x ‘2’ matrix with all the calculated values from the set.

Examples

#Generate a 2D sample:
points2d = generate_pmj02_set(n=1000)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

#Generate a longer sequence of values from that set
points2d = generate_pmj02_set(n=1500)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

#Generate a new set by changing the seed
points2d = generate_pmj02_set(n=1500, seed=10)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

#Integrate the value of pi by counting the number of randomly generated points that fall
#within the unit circle.
pointset = generate_pmj02_set(10000)

pi_estimate = 4*sum(pointset[,1] * pointset[,1] + pointset[,2] * pointset[,2] < 1)/10000
pi_estimate
generate_pmjbn_set  
**Generate 2D Progressive Multi-Jittered (with blue noise) Set**

---

**Description**

Generate a set of values from a Progressive Multi-Jittered (with blue noise) set.

**Usage**

`generate_pmjbn_set(n, seed = 0)`

**Arguments**

- **n**: The number of 2D values to extract.
- **seed**: Default `0`. The random seed.

**Value**

An `n` x `2` matrix with all the calculated values from the set.

**Examples**

```r
# Generate a 2D sample:
points2d = generate_pmjbn_set(n=1000)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

# Generate a longer sequence of values from that set
points2d = generate_pmjbn_set(n=1500)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

# Generate a new set by changing the seed
points2d = generate_pmjbn_set(n=1500, seed=10)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

# Integrate the value of pi by counting the number of randomly generated points that fall within the unit circle.
pointset = generate_pmjbn_set(10000)

pi_estimate = 4*sum(pointset[,1] * pointset[,1] + pointset[,2] * pointset[,2] < 1)/10000
pi_estimate
```
generate pmj set

**Description**

Generate a set of values from a Progressive Multi-Jittered set.

**Usage**

```r
generate_pmj_set(n, seed = 0)
```

**Arguments**

- `n`: The number of 2D values to extract.
- `seed`: Default ‘0’. The random seed.

**Value**

An ‘n’ x ‘2’ matrix with all the calculated values from the set.

**Examples**

```r
#Generate a 2D sample:
points2d = generate_pmj_set(n=1000)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

#Generate a longer sequence of values from that set
points2d = generate_pmj_set(n=1500)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

#Generate a new set by changing the seed
points2d = generate_pmj_set(n=1500, seed=10)
plot(points2d, xlim=c(0,1), ylim=c(0,1))

#Integrate the value of pi by counting the number of randomly generated points that fall
#within the unit circle.
pointset = generate_pmj_set(10000)

pi_estimate = 4*sum(pointset[,1] * pointset[,1] + pointset[,2] * pointset[,2] < 1)/10000
pi_estimate
```
**generate_sobol_owen_set**

Generate Owen-scrambled Sobol Set

**Description**

Generate a set of values from an Owen-scrambled Sobol set.

**Usage**

`generate_sobol_owen_set(n, dim, seed = 0)`

**Arguments**

- **n**
  - The number of values (per dimension) to extract.

- **dim**
  - The number of dimensions of the sequence.

- **seed**
  - Default ‘0’. The random seed.

**Value**

An ‘n’ x ‘dim’ matrix with all the calculated values from the set.

**Examples**

```r
#Generate a 2D sample:
points2d = generate_sobol_owen_set(n=1000, dim = 2)
plot(points2d, xlim=c(0,1),ylim=c(0,1))

#Generate a longer sequence of values from that set
points2d = generate_sobol_owen_set(n=1500, dim = 2)
plot(points2d, xlim=c(0,1),ylim=c(0,1))

#Integrate the value of pi by counting the number of randomly generated points that fall within the unit circle.
pointset = matrix(generate_sobol_owen_set(n=10000,dim=2),ncol=2)
pi_estimate = 4*sum(pointset[,1] * pointset[,1] + pointset[,2] * pointset[,2] < 1)/10000
pi_estimate
```
generate_sobol_set

Description
Generate a set of values from a Sobol set.

Usage
generate_sobol_set(n, dim, seed = 0)

Arguments

- \( n \) The number of values (per dimension) to extract.
- \( \text{dim} \) The number of dimensions of the sequence.
- \( \text{seed} \) Default ‘0’: The random seed.

Value
A single numeric value representing the ‘\( i \)'th element in the ‘\( \text{dim} \)' dimension.

Examples

```r
#Generate a 2D sample:
points2d = generate_sobol_set(n=1000, dim = 2)
plot(points2d, xlim=c(0,1),ylim=c(0,1))

#Generate a longer sequence of values from that set
points2d = generate_sobol_set(n=1500, dim = 2)
plot(points2d, xlim=c(0,1),ylim=c(0,1))

#Integrate the value of pi by counting the number of randomly generated points that fall within the unit circle.
pointset = matrix(generate_sobol_set(10000,dim=2),ncol=2)

pi_estimate = 4*sum(pointset[,1] * pointset[,1] + pointset[,2] * pointset[,2] < 1)/10000
pi_estimate
```
Index

generate_halton_faure_set, 2
generate_halton_faure_single, 3
generate_halton_random_set, 3
generate_halton_random_single, 4
generate_pj_set, 5
generate_pmj02_set, 7
generate_pmj02bn_set, 6
generate_pmj_set, 9
generate_pmjbn_set, 8
generate_sobol_owen_set, 10
generate_sobol_set, 11