Package ‘DiceDesign’

December 7, 2023

Type  Package
Title  Designs of Computer Experiments
Version 1.10
Date  2023-11-30
Author  Jessica Franco, Delphine Dupuy, Olivier Roustant,  
        Patrice Kiener, Guillaume Damblin and Bertrand Iooss.
Maintainer  Celine Helbert <Celine.Helbert@ec-lyon.fr>
Description  Space-Filling Designs and space-filling criteria (distance-based and uniformity-based), with emphasis to computer experiments; <doi:10.18637/jss.v065.i11>.
License  GPL-3
Depends  R (>= 2.10)
Suggests  rgl, randtoolbox, lattice
Encoding  UTF-8
NeedsCompilation  yes
LazyData  true
Repository  CRAN
Date/Publication  2023-12-07 12:00:10 UTC

R topics documented:

DiceDesign-package ......................................................... 2  
coverage ............................................................... 4  
discrepancyCriteria ....................................................... 6  
discrepESE_LHS ......................................................... 7  
discrepSA_LHS .......................................................... 9  
dmaxDesign ............................................................. 11  
factDesign ............................................................. 13  
faureprimeDesign ....................................................... 14  
lhsDesign .............................................................. 16  
maximinESE_LHS ........................................................ 17  
maximinSA_LHS ........................................................ 19
Description

Space-Filling Designs (SFD) and space-filling criteria (distance-based and uniformity-based).

Details

This package provides tools to create some specific Space-Filling Design (SFD) and to test their quality:

- Latin Hypercube designs (randomized or centered)
- Strauss SFD and Maximum entropy SFD, WSP designs
- Optimal (low-discrepancy and maximin) Latin Hypercube designs by simulated annealing and genetic algorithms,
- Orthogonal and Nearly Orthogonal Latin Hypercube designs,
- Discrepancies criteria, distance measures,
- Minimal spanning tree criteria,
- Radial scanning statistic
Note

Part of this work was conducted on 2006-2009 within the frame of the DICE (Deep Inside Computer Experiments) Consortium between ARMINES, Renault, EDF, IRSN, ONERA and TOTAL S.A. (http://dice.emse.fr/).

In this package, only Faure’s sequence is implemented. Note that the randtoolbox package provides the following quasi random sequences: the Sobol sequence, the Halton (hence Van Der Corput) sequence and the Torus sequence (also known as Kronecker sequence). Note also that the lhs package provides other types of algorithms to compute optimized LHS.

Author(s)


Maintainer: Celine Helbert <Celine.Helbert@ec-lyon.fr>

References


Examples

# ************************************
# Designs of experiments
# ************************************
# A maximum entropy design with 20 points in \([0,1]^2\)
\[\texttt{p} \leftarrow \texttt{dmaxDesign(20,2,0.9,200)}\]
\[\texttt{plot(p$design,xlim=c(0,1),ylim=c(0,1))}\]

# Change the dimnames, adjust to range (-10, 10) and round to 2 digits
\[\texttt{xDRDN(p, letter = "T", dgts = 2, range = c(-10, 10))}\]

# ************************
# Criteria: L2-discrepancy
# ************************
\[\texttt{dp} \leftarrow \texttt{discrepancyCriteria(p$design, type=c('L2', 'C2'))}\]

# Coverage measure
\[\texttt{covp} \leftarrow \texttt{coverage(p$design)}\]

# ************************
# Criteria: Minimal Spanning Tree
# ************************
\[\texttt{mstCriteria(p$design, plot2d=TRUE)}\]

# Radial scanning statistic: Detection of defects of Sobol designs
# *******************************************************************************
# requires randtoolbox package
\[\texttt{library(randtoolbox)}\]

# in 2D
\[\texttt{rss} \leftarrow \texttt{rss2d(design=sobol(n=20, dim=2), lower=c(0,0), upper=c(1,1), type="l", col="red")}\]

# in 8D. All pairs of dimensions are tried to detect the worst defect
# (according to the specified goodness-of-fit statistic).
\[\texttt{d} \leftarrow 8\]
\[\texttt{n} \leftarrow 10*d\]
\[\texttt{rss} \leftarrow \texttt{rss2d(design=sobol(n=n, dim=d), lower=rep(0,d), upper=rep(1,d), type="l", col="red")}\]

# avoid this defect with scrambling ?
# 1. Faure-Tezuka scrambling (type "sobol" for more details and options)
\[\texttt{rss} \leftarrow \texttt{rss2d(design=sobol(n=n, dim=d, scrambling=2), lower=rep(0,d), upper=rep(1,d), type="l", col="red")}\]
# 2. Owen scrambling
\[\texttt{rss} \leftarrow \texttt{rss2d(design=sobol(n=n, dim=d, scrambling=1), lower=rep(0,d), upper=rep(1,d), type="l", col="red")}\]

## coverage

<table>
<thead>
<tr>
<th>Coverage</th>
</tr>
</thead>
</table>

### Description

Compute the coverage measure
coverage

Usage

coverage(design)

Arguments

design a matrix (or a data.frame) representing the design of experiments representing the design of experiments in the unit cube \([0,1]^d\). If this last condition is not fulfilled, a transformation into \([0,1]^d\) is applied before the computation of the criteria.

Details

The coverage criterion is defined by

\[
\text{coverage} = \frac{1}{\bar{\gamma}} \left[ \frac{1}{n} \sum_{i=1}^{n} (\gamma_i - \bar{\gamma})^2 \right]^{1/2}
\]

where \(\gamma_i\) is the minimal distance between the point \(x_i\) and the other points of the design and \(\bar{\gamma}\) is the mean of the \(\gamma_i\).

Note that for a regular mesh, \(\text{cov}=0\). Then, a small value of cov means that the design is close to a regular grid.

Value

A real number equal to the value of the coverage criterion for the design.

Author(s)

J. Franco

References


See Also

other distance criteria like meshRatio, phiP and mindist.
discrepancy measures provided by discrepancyCriteria.

Examples

dimension <- 2
n <- 40
X <- matrix(runif(n*dimension), n, dimension)
coverage(X)
discrepancyCriteria

Discrepancy measure

Description

Compute discrepancy criteria.

Usage

discrepancyCriteria(design,type='all')

Arguments

design  a matrix (or a data.frame) corresponding to the design of experiments. The discrepancy criteria are computed for a design in the unit cube \([0,1]^d\). If this condition is not satisfied the design is automatically rescaled.

type  type of discrepancies (single value or vector) to be computed:

- 'all'  all type of discrepancies (default)
- 'C2'  centered L2-discrepancy
- 'L2'  L2-discrepancy
- 'L2star'  L2star-discrepancy
- 'M2'  modified L2-discrepancy
- 'S2'  symmetric L2-discrepancy
- 'W2'  wrap-around L2-discrepancy
- 'Mix2'  mixture L2-discrepancy

Details

The discrepancy measures how far a given distribution of points deviates from a perfectly uniform one. Different L2 discrepancies are available in DiceDesign. For example, if we denote by \(Vol(J)\) the volume of a subset \(J\) of \([0;1]^d\) and \(A(X;J)\) the number of points of \(X\) falling in \(J\), the L2 discrepancy is:

\[
D_{L2}(X) = \left[ \int_{[0,1]^{2d}} \left( \frac{A(X,J_{a,b})}{n} - Vol(J_{a,b}) \right)^2 dadb \right]^{1/2}
\]

where \(a = (a_1;...;a_d)'\), \(b = (b_1;...;b_d)'\) and \(J_{a,b} = [a_1;b_1) \times ... \times [a_d;b_d)\). The other L2-discrepancies are defined according to the same principle with different form from the subset \(J\). Among all the possibilities, discrepancyCriteria implements only the L2 discrepancies because it can be expressed analytically even for high dimension.

Centered L2-discrepancy is computed using the analytical expression done by Hickernell (1998). The user will refer to Pleming and Manteufel (2005) to have more details about the wrap around discrepancy.
**Value**

A list containing the L2-discrepancies of the design.

**Author(s)**

J. Franco, D. Dupuy & B. Iooss

**References**


**See Also**

distance criteria (coverage, meshRatio, mindist and phiP)

**Examples**

```r
dimension <- 2
n <- 40
X <- matrix(runif(n*dimension), n, dimension)
discrepancyCriteria(X)
```

**Description**

The objective is to produce low-discrepancy LHS. ESE is a powerful genetic algorithm to produce space-filling designs. It has been adapted here to main discrepancy criteria.

**Usage**

```r
discrepESE_LHS(design, T0=0.005*discrepancyCriteria(design,type='Var')[[1]], inner_it=100, J=50, it=2, criterion="C2")
```
Arguments

design  a matrix (or a data.frame) corresponding to the design of experiments.

T0       The initial temperature of the ESE algorithm

inner_it The number of iterations for inner loop

J        The number of new proposed LHS inside the inner loop

it       The number of iterations for outer loop

criterion The criterion to be optimized. One can choose three different L2-discrepancies:
           the C2 (centered) discrepancy ("C2"), the L2-star discrepancy ("L2star") and the
           W2 (wrap-around) discrepancy ("W2")

Details

This function implements a stochastic algorithm (ESE) to produce optimized LHS. It is based on
Jin et al works (2005). Here, it has been adapted to some discrepancy criteria taking into account
new ideas about the revaluations of discrepancy value after a LHS elementary perturbation (in order
to avoid computing all terms in the discrepancy formulas).

Value

A list containing:

InitialDesign the starting design

T0        the initial temperature of the ESE algorithm

inner_it the number of iterations for inner loop

J         the number of new proposed LHS inside the inner loop

it        the number of iterations for outer loop

criterion the criterion to be optimized

design    the matrix of the final design (low-discrepancy LHS)
critValues vector of criterion values along the iterations
tempValues vector of temperature values along the iterations
probValues vector of acceptance probability values along the iterations

Author(s)

G.Damblin & B. Iooss

References


discrepSA_LHS

See Also
Latin Hypercube Sample({\texttt{lhsDesign}}), discrepancy criteria({\texttt{discrepancyCriteria}}), geometric criterion (\texttt{mindistphiP}), optimization ({\texttt{maximinSA_LHS, maximinESE_LHS, discrepSA_LHS}})

Examples

```r
## Not run:
dimension <- 2
n <- 10
X <- lhsDesign(n, dimension)$design
Xopt <- discrepESE_LHS(X, T0=0.005*discrepancyCriteria(X, type='C2')[[1]],
                      inner_it=100, J=50, it=2)
plot(Xopt$design)
plot(Xopt$critValues, type="l")
## End(Not run)
```

---

discrepSA_LHS

Simulated annealing (SA) routine for Latin Hypercube Sample (LHS) optimization via L2-discrepancy criteria

Description

The objective is to produce low-discrepancy LHS. SA is an efficient algorithm to produce space-filling designs. It has been adapted here to main discrepancy criteria.

Usage

discrepSA_LHS(design, T0=10, c=0.95, it=2000, criterion="C2", profile="GEOM", Imax=100)

Arguments

design a matrix (or a data.frame) corresponding to the design of experiments
T0 The initial temperature
c A constant parameter regulating how the temperature goes down
it The number of iterations
criterion The criterion to be optimized. One can choose three different L2-discrepancies: the C2 (centered) discrepancy ("C2"), the L2-star discrepancy ("L2star") and the W2 (wrap-around) discrepancy ("W2")
profile The temperature down-profile, purely geometric called "GEOM", geometrical according to the Morris algorithm called "GEOM_MORRIS" or purely linear called "LINEAR"
Imax A parameter given only if you choose the Morris down-profile. It adjusts the number of iterations without improvement before a new elementary perturbation
Details

This function implements a classical routine to produce optimized LHS. It is based on the work of Morris and Mitchell (1995). They have proposed a SA version for LHS optimization according to mindist criterion. Here, it has been adapted to some discrepancy criteria taking in account new ideas about the reevaluations of a discrepancy value after a LHS elementary perturbation (in order to avoid computing all terms in the discrepancy formulas).

Value

A list containing:

- **InitialDesign**: the starting design
- **T0**: the initial temperature of the SA algorithm
- **c**: the constant parameter regulating how the temperature goes down
- **it**: the number of iterations
- **criterion**: the criterion to be optimized
- **profile**: the temperature down-profile
- **Imax**: The parameter given in the Morris down-profile
- **design**: the matrix of the final design (low-discrepancy LHS)
- **critValues**: vector of criterion values along the iterations
- **tempValues**: vector of temperature values along the iterations
- **probaValues**: vector of acceptation probability values along the iterations

Author(s)

G. Damblin & B. Iooss

References


See Also

Latin Hypercube Sample(*lhsDesign*), discrepancy criteria(*discrepancyCriteria*), geometric criterion (*mindistphiP*), optimization (*maximinSA_LHS*, *maximinESE_LHS*, *discrepESE_LHS*)
Examples

dimension <- 2
n <- 10
X <- lhsDesign(n, dimension)$design

## Optimize the LHS with C2 criterion
Xopt <- discrepSA_LHS(X, T0=10, c=0.99, it=2000, criterion="C2")
plot(Xopt$design)
plot(Xopt$critValues, type="l")

## Optimize the LHS with C2 criterion and GEOM_MORRIS profile
## Not run:
Xopt2 <- discrepSA_LHS(X, T0=10, c=0.99, it=1000, criterion="C2", profile="GEOM_MORRIS")
plot(Xopt2$design)
## End(Not run)

dmaxDesign

Maximum Entropy Designs

Description

Space-Filling Designs with \( n \) experiments based on covariance matrix in \([0,1]^d\).

Usage

dmaxDesign(n, dimension, range, niter_max=1000, seed=NULL)

Arguments

- \( n \): number of experiments
- \( \text{dimension} \): number of variables
- \( \text{range} \): range of variogram
- \( \text{niter\_max} \): number of iterations
- \( \text{seed} \): seed used to generate uniform design

Details

Maximum entropy design is a kind of optimal design based on Shannon’s definition of entropy as the amount of information. Originally, maximum entropy sampling was proposed by Shewry and Wynn (1987). The goal of the design is to maximize the entropy defined as the determinant of the correlation matrix using a Fedorov-Mitchell exchange algorithm.

The spatial correlation matrix is defined by \( C = (\rho_{ij}) \):

\[
\rho_{ij} = \begin{cases} 
1 - \gamma(h_{ij}) & \text{if } h_{ij} \leq a, \\
0 & \text{if } h_{ij} > a,
\end{cases}
\]
where \( h_{ij} \) is the distance between \( x_i \) and \( x_j \), \( a \) denotes the range of the variogram and \( \gamma \) is a spherical variogram:

\[
\gamma(h) = 1.5 \frac{h}{a} - 0.5 \left( \frac{h}{a} \right)^3 \quad \text{for } h \leq a
\]

**Value**

A list with components:

- \( n \) the number of points
- \( \text{design} \) the design of experiments
- \( \text{dimension} \) the number of variables
- \( \text{range} \) the range of the variogram
- \( \text{niter}_\text{mx} \) the number of iterations
- \( \text{design}_\text{init} \) the initial distribution
- \( \text{det}_\text{init} \) the value of the determinant for the initial distribution
- \( \text{det}_\text{end} \) the value of the determinant at the end of the procedure
- \( \text{seed} \) the value of the seed

**Author(s)**

J. Franco

**References**


**Examples**

```r
n <- 20
dimension <- 2
range <- 0.9
niter_max <- 200
out <- dmaxDesign(n, dimension, range, niter_max)

## Change the dimnames, adjust to range (-10, 10) and round to 2 digits
xDRDN(out, letter = "T", dgts = 2, range = c(-10, 10))
```
FactDesign

Full Factorial Designs

Description

Create a factorial design with \( n = \text{pow(levels, dimension)} \) experiments in \([0,1]^d\).

Usage

\text{factDesign(dimension, levels)}

Arguments

dimension \quad \text{an integer given the number of input variables}
levels \quad \text{an integer given the number of levels}

Details

It is possible to take a different number of levels for any factor. In this case, the argument levels should be a vector.

Value

\text{factDesign} \quad \text{returns a list containing all the input arguments detailed before, plus the following components:}

\text{n} \quad \text{the number of experiments}
\text{design} \quad \text{the design of experiments}

Author(s)

G. Pujol and J. Franco

Examples

```r
## First example
g1 <- factDesign(2, 7)
plot(g1$design, xlim=c(0,1), ylim=c(0,1))

## Second example
g2 <- factDesign(2, c(2,7))
plot(g2$design, xlim=c(0,1), ylim=c(0,1))

## Change the dimnames, adjust to range (-10, 10) and round to 2 digits
xORDN(g1, letter = "T", dgts = 2, range = c(-10, 10))
xORDN(g2, letter = "T", dgts = 2, range = c(-10, 10))
```
faureprimeDesign is a special case of the low discrepancy Faure sequence

Description

Generate a Faure sequence with \( n = p^u - 1 \) experiments in \([0,1]^d\) or other domains (see the details) where \( p \) is the first prime number equal or larger than \( d \) and \( u \) is an exponent, usually 2.

Usage

```r
faureprimeDesign(dimension, u = 2, range = c(0, -1))
```

Arguments

- `dimension`: the number of variables (< 199)
- `u`: the exponent applied to the prime number
- `range`: the scale (min and max) of the inputs. See the details for the six predefined ranges.

Details

This is a special case of `runif.faure` where the number of generated points depends exclusively on the dimension and the selected exponent. For the exponent \( u = 2 \), the design is orthogonal and has resolution 4. It is a perfect grid \((p - 1)(p + 1)\) on each pair of variables where \( p \) is the first prime number equal or larger than the dimension \( d \).

Six domain ranges are predefined and cover most applications:

- \( c(0, 0) \) corresponds to \([0, n]^d\).
- \( c(1, 1) \) corresponds to \([1 - n, n - 1]^d = [2 - p^u, p^u - 2]^d\).
- \( c(0, 1) \) corresponds to \([0, 1]^d\).
- \( c(0,-1) \) corresponds to \([p^{-u}, 1 - p^{-u}]^d\).
- \( c(-1,-1) \) corresponds to \([-1 + 2p^{-u}, 1 - 2p^{-u}]^d\).
- \( c(-1, 1) \) corresponds to \([-1,1]^d\).

Value

`faureprimeDesign` returns a list with the following components:

- `design`: the design of experiments
- `n`: the number of experiments
- `dimension`: the dimension
- `prime`: the prime number
- `u`: the exponent
faureprimeDesign

Author(s)

P. Kiener

References


Examples

```r
## Range c(0,-1) returns the design produced by runif.faure()
plan1 <- runif.faure(n = 24, dimension = 5)$design; plan1
plan2 <- faureprimeDesign(dimension = 5, range = c(0,-1))$design; plan2
all.equal(plan1, plan2, tolerance = 1e-15)

## Range c(0,0) returns the original sequence of integers.
## The first (p-1) lines are on the first diagonal.
## The remaining lines are LHSs grouped in p-1 blocks of p rows.
d <- p <- 5
plan <- faureprimeDesign(dimension = d, range = c(0,0))$design; plan
apply(plan, 2, sort)

## A regular grid (p-1)x(p+1) rotated by a small angle
pairs(plan)

plot(plan[,1], plan[,2], las = 1)
points(plan[1:(p-1),1], plan[1:(p-1),2], pch = 17, cex = 1.6)
abline(v = plan[1:(p-1),1], col = 4)

## Designs of dimensions 24x5 in various ranges
lstrg <- list(p0p0 = c(0,0), p1p1 = c(1,1), p0p1 = c(0,1), p0m1 = c(0,-1), m1m1 = c(-1,-1), m1p1 = c(-1,1))
lst <- lapply(lstrg, function(rg) faureprimeDesign(dimension = 5, u = 2, range = rg)$design)
lapply(lst, tail)
sapply(lst, range)

## The odd designs (p1m1, m1m1, m1p1) are orthogonal and have resolution 4.
library(lattice)

mat <- lst$m1m1; colnames(mat) <- LETTERS[1:5]
fm1 <- ~ (A+B+C+D+E)^2+I(A^2)+I(B^2)+I(C^2)+I(D^2)+I(E^2)
mmm <- model.matrix(fm1, data = as.data.frame(mat))[,,-1]; tail(mmm)
cmm <- round(cov2cor(crossprod(mmm)), 3); cmm
lattice::levelplot(cmm[, ncol(cmm):1], at = seq(-1, 1, length.out = 10),
                   col.regions = rev(grDevices::hcl.colors(9, "PuOr")))
```
lhsDesign

Latin Hypercube Designs

Description

Simple (random) Latin Hypercube Design (randomized or centered) with \( n \) experiments in \([0,1]^d\).

Usage

lhsDesign(n, dimension, randomized=TRUE, seed=NULL)

Arguments

- \( n \) : number of experiments
- \( \text{dimension} \) : number of variables
- \( \text{randomized} \) : TRUE for randomized LHS; FALSE for centered LHS
- \( \text{seed} \) : seed used to generate the random permutations and perturbations

Details

This program builds a Latin Hypercube Design (LHD), also called a Latin Hypercube Sample (LHS), on the space \([0,1]^d\) (with uniform probability measures). LHD aims at ensuring that each variable has its whole range well scanned: the range of each variable is divided into \( n \) equally probable stratas. Each stratum of each variable contains only one point of the LHD. Centered LHD is obtained by choosing for each point the center of the corresponding case, while randomized LHD is obtained by adding random perturbations inside each point case.

Once the sample is generated, the uniform sample from a column can be transformed to any distribution by using the quantile functions.

Value

A list with components:

- \( n \) : the number of points
- \( \text{dimension} \) : the number of variables
- \( \text{design} \) : the design of experiments
- \( \text{randomized} \) : the type of LHD
- \( \text{seed} \) : the value of the seed

Author(s)

B. Iooss
maximinESE_LHS

References

McKay M., Conover W. and Beckman R. (1979) A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics, 21, 2, 239-245.


See Also

LHD optimization (maximinSA_LHS, discrepSA_LHS, maximinESE_LHS, discrepESE_LHS)

Examples

n <- 20
dimension <- 2
out <- lhsDesign(n, dimension)
out$design

## Change the dimnames, adjust to range (-10, 10) and round to 2 digits
xORDN(out, letter = "T", dgts = 2, range = c(-10, 10))

maximinESE_LHS

Enhanced Stochastic Evolutionary (ESE) algorithm for Latin Hypercube Sample (LHS) optimization via phiP criteria

Description

The objective is to produce maximin LHS. ESE is a powerful genetic algorithm allowing to produce space-filling designs.

Usage

maximinESE_LHS(design, T0=0.005*phiP(design,p=50), inner_it=100, J=50, it=1, p=50)

Arguments

design a matrix (or a data.frame) corresponding to the design of experiments.

T0 The initial temperature of the ESE algorithm

inner_it The number of iterations for inner loop

J The number of new proposed LHS inside the inner loop

it The number of iterations for outer loop

p power required in phiP criterion

Details

This function implements a stochastic algorithm (ESE) to produce optimized LHS. It is based on Jin et al works (2005).
maximinESE_LHS

Value

A list containing:

- `InitialDesign` the starting design
- `T0` the initial temperature of the ESE algorithm
- `inner_it` the number of iterations for inner loop
- `J` the number of new proposed LHS inside the inner loop
- `it` the number of iterations for outer loop
- `p` power required in phiP criterion
- `design` the matrix of the final design (maximin LHS)
- `critValues` vector of criterion values along the iterations
- `tempValues` vector of temperature values along the iterations
- `probaValues` vector of acceptance probability values along the iterations

Author(s)

G. Damblin & B. Iooss

References


See Also

Latin Hypercube Sample (`lhsDesign`), discrepancy criteria (`discrepancyCriteria`), geometric criterion (`mindist`, `phiP`), optimization (`maximinSA_LHS`, `discrepESE_LHS`, `discrepSA_LHS`)

Examples

```r
dimension <- 2
n <- 10
X <- lhsDesign(n, dimension)$design
Xopt <- maximinESE_LHS(X, T0=0.005*phiP(X), inner_it=100, J=50, it=2)
plot(Xopt$design)
plot(Xopt$critValues, type="l")
```
maximinSA_LHS

Simulated annealing (SA) routine for Latin Hypercube Sample (LHS) optimization via phiP criteria

Description

The objective is to produce maximin LHS. SA is an efficient algorithm to produce space-filling designs.

Usage

maximinSA_LHS(design, T0=10, c=0.95, it=2000, p=50, profile="GEOM", Imax=100)

Arguments

design  a matrix (or a data.frame) corresponding to the design of experiments
T0      The initial temperature of the SA algorithm
C       A constant parameter regulating how the temperature goes down
it      The number of iterations
p       power required in phiP criterion
profile The temperature down-profile, purely geometric called "GEOM", geometrical according to the Morris algorithm called "GEOM_MORRIS" or purely linear called "LINEAR"
Imax    A parameter given only if you choose the Morris down-profile. It adjusts the number of iterations without improvement before a new elementary perturbation

Details

This function implements a classical routine to produce optimized LHS. It is based on the work of Morris and Mitchell (1995). They have proposed a SA version for LHS optimization according to mindist criterion. Here, it has been adapted to the phiP criterion. It has been shown (Pronzato and Muller, 2012, Damblin et al., 2013) that optimizing phiP is more efficient to produce maximin designs than optimizing mindist. When \( p \) tends to infinity, optimizing a design with \( \phi_p \) is equivalent to optimizing a design with mindist.

Value

A list containing:

InitialDesign  the starting design
T0            the initial temperature of the SA algorithm
C             the constant parameter regulating how the temperature goes down
it           the number of iterations
p             power required in phiP criterion
profile  the temperature down-profile
Imax    The parameter given in the Morris down-profile
design  the matrix of the final design (maximin LHS)
critValues  vector of criterion values along the iterations
tempValues  vector of temperature values along the iterations
probaValues  vector of acceptance probability values along the iterations

Author(s)
G. Damblin & B. Iooss

References


See Also
Latin Hypercube Sample (*lhsDesign*), discrepancy criteria (*discrepancyCriteria*), geometric criterion (*mindist, phiP*), optimization (*discrepSA_LHS, maximinESE_LHS, discrepESE_LHS*)

Examples

dimension <- 2
n <- 10
X <- lhsDesign(n, dimension)$design
Xopt <- maximinSA_LHS(X, T0=10, c=0.99, it=2000)
plot(Xopt$design)
plot(Xopt$critValues, type="l")
plot(Xopt$tempValues, type="l")

## Not run:
Xopt <- maximinSA_LHS(X, T0=10, c=0.99, it=1000, profile="GEOM_MORRIS")

## End(Not run)
### Description

The `meshRatio` criterion is the ratio between the maximum and the minimum distance between two points of the experimental design.

### Usage

```r
meshRatio(design)
```

### Arguments

- **design**: a matrix (or a data.frame) representing the design of experiments in the unit cube \([0,1]^d\). If this last condition is not fulfilled, a transformation into \([0,1]^d\) is applied before the computation of the criteria.

### Details

The `meshRatio` criterion is defined by

\[
meshRatio = \frac{\max_{1 \leq i \leq n} \gamma_i}{\min_{1 \leq i \leq n} \gamma_i}
\]

where \(\gamma_i\) denotes the minimal distance between the point \(x_i\) and the other points of the design.

Note that for a regular mesh, `meshRatio`=1.

### Value

A real number equal to the value of the `meshRatio` criterion for the design.

### Author(s)

J. Franco

### References

Gunzburger M. and Burdick J. (2004), Uniformity measures for point samples in hypercubes, [https://people.sc.fsu.edu/~jburkardt/](https://people.sc.fsu.edu/~jburkardt/).

### See Also

Other distance criteria like `meshRatio`, `phiP` and `mindist`. Discrepancy measures provided by `discrepancyCriteria`. 
Examples

dimension <- 2
n <- 40
X <- matrix(runif(n*dimension), n, dimension)
meshRatio(X)

mindist

Description

Compute the mindist criterion (also called maximin)

Usage

mindist(design)

Arguments

design a matrix (or a data.frame) representing the design of experiments in the unit cube \([0,1]^d\). If this last condition is not fulfilled, a transformation into \([0,1]^d\) is applied before the computation of the criteria.

Details

The mindist criterion is defined by

\[
\text{mindist} = \min_{x_i \in X} (\gamma_i)
\]

where \(\gamma_i\) is the minimal distance between the point \(x_i\) and the other points \(x_k\) of the design.

A higher value corresponds to a more regular scattering of design points.

Value

A real number equal to the value of the mindist criterion for the design.

Author(s)

J. Franco

References

Gunzburner M., Burkdart J. (2004), Uniformity measures for point samples in hypercubes, https://people.sc.fsu.edu/~jburkardt/.


See Also

other distance criteria like meshRatio and phiP, discrepancy measures provided by discrepancyCriteria.

Examples

dimension <- 2
n <- 40
X <- matrix(runif(n*dimension), n, dimension)
mindist(X)

mstCriteria derivative

Description

Compute both the mean and the standard deviation of the Minimal Spanning Tree (MST)

Usage

mstCriteria(design, plot2d="FALSE")

Arguments

  design a matrix (or a data.frame) corresponding to the design of experiments.
  plot2d an argument for visualizing the mst of a 2d design

Details

In our context, a MST is a tree whose the sum of the lengths of the edges is minimal. Even if unicity does not hold, the overall length is stable. The mean and the standard deviation of the lengths of the edges are usually derived to analyze the geometric profile of the design. A large mean and a small standard deviation characterize a so-called quasi-periodic design.

Value

A list containing two components:

tree a list containing the MST: each component of it contains a vector with all vertices which are connected with the experiment corresponding to the number of the components
stats vector with both the mean and the standard deviation values of the lengths of the edges

Author(s)

G. Damblin & B. Iooss
References


Examples

```r
dimension <- 2
n <- 40
X <- matrix(runif(n*dimension), n, dimension)
mstCriteria(X, plot2d=TRUE)
```

---

**nolhDesign**

*Cioppa’s Nearly Orthogonal Latin Hypercube Designs*

**Description**

This function generates a NOLH design of dimension 2 to 29 and normalizes it to the selected range. The design is extracted from Cioppa’s NOLHdesigns list.

**Usage**

```r
nolhDesign(dimension, range = c(0, 1))
```

**Arguments**

- `dimension`: number of input variables
- `range`: the scale (min and max) of the inputs. Range (0, 0) and (1, 1) are special cases and call integer ranges \((-m, m)\) and \((0, 2m)\). See the examples

**Value**

A list with components:

- `n`: the number of lines/experiments
- `dimension`: the number of columns/input variables
- `design`: the design of experiments
Author(s)


See Also

Cioppa’s list NOLHdesigns. Other NOLH and OLH designs: nolhdrDesign, olhDesign.

Examples

```r
## Classical normalizations
nolhDesign(8, range = c(1, 1))
nolhDesign(8, range = c(0, 0))
nolhDesign(8, range = c(0, 1))
nolhDesign(8, range = c(-1, 1))

## Change the dimnames, adjust to range (-10, 10) and round to 2 digits
xDRDN(nolhDesign(8), letter = "T", dgts = 2, range = c(-10, 10))

## A list of designs
lapply(5:9, function(n) nolhDesign(n, range = c(-1, 1))$design)
```

NOLHdesigns

List of Cioppa’s Nearly Orthogonal Latin Hypercubes designs

Description

A list of the NOLH designs for 2 to 29 input variables proposed by Cioppa in 2007. These designs combine a latin structure, orthogonality between the main terms and the interactions (+ squares) and reduced correlations between the interactions (+ squares).

This list combines the Excel spreadsheets published by Sanchez (see Source). It is used internally by the function nolhDesign which provides various normalizations.

Usage

NOLHdesigns

Format

A list of 5 matrices representing designs of experiments for 8 to 29 input variables:

- nolh2_7: 2 to 7 input variables, 17 experiments.
- nolh8_11: 8 to 11 input variables, 33 experiments.
- nolh12_16: 12 to 16 input variables, 65 experiments.
- nolh17_22: 17 to 22 input variables, 129 experiments.
- nolh23_29: 23 to 29 input variables, 257 experiments.
De Rainville’s Nearly Orthogonal Latin Hypercube Designs

This function generates a NOLH design of dimension 2 to 29 and normalizes it to the selected range. From 2 to 7 input variables, the design is extracted from Cioppa’s NOLHdesigns list and from 8 to 29 input variables it is extracted from De Rainville’s NOLHDRdesigns list.

Usage

nolhdrDesign(dimension, range = c(0, 1))
Arguments

- **dimension**: number of input variables
- **range**: the scale (min and max) of the inputs. Range (0, 0) and (1, 1) are special cases and call integer ranges \((-m, m)\) and \((0, 2m)\). See the examples.

Value

A list with components:

- **n**: the number of lines/experiments
- **dimension**: the number of columns/input variables
- **design**: the design of experiments

Author(s)


See Also

De Rainville’s list `NOLHDRdesigns`. Other NOLH or OLH designs: `nolhDesign`, `olhDesign`.

Examples

```r
## Classical normalizations
nolhdrDesign(8, range = c(1, 1))
nolhdrDesign(8, range = c(0, 1))
nolhdrDesign(8, range = c(0, 0))
nolhdrDesign(8, range = c(-1, 1))

## Change the dimnames, adjust to range (-10, 10) and round to 2 digits
xDRDN(nolhdrDesign(8), letter = "T", dgts = 2, range = c(-10, 10))

## A list of designs
lapply(5:9, function(n) nolhdrDesign(n, range = c(-1, 1))$design)
```

### NOLHDRdesigns

**List of De Rainville’s Nearly Orthogonal Latin Hypercubes designs**

Description

A list of the NOLH designs for 8 to 29 input variables proposed by De Rainville in 2012. These designs are said to be an improvement of Cioppa’s NOLH designs as they have the same structure but better dispersion measures like the discrepancy.

This list combines the csv files published by De Rainville (see Source), centered and normalized to integer values. It is used internally by the function `nolhdrDesign` which provides various normalizations.
Usage

NOLHDRdesigns

Format

A list of 22 matrices representing designs of experiments for 8 to 29 input variables:

- `nolhdr08` to `nolhdr11`: 8, 9, 10, 11 input variables, 33 experiments.
- `nolhdr12` to `nolhdr16`: 12, 13, 14, 15, 16 input variables, 65 experiments.
- `nolhdr17` to `nolhdr22`: 17, 18, 19, 20, 21, 22 input variables, 129 experiments.
- `nolhdr23` to `nolhdr29`: 23, 24, 25, 26, 27, 28, 29 input variables, 257 experiments.

Author(s)

F.-M. De Rainville for the designs. P. Kiener for the R code.

Source

Main website: [http://qrand.gel.ulaval.ca/](http://qrand.gel.ulaval.ca/)
The python source code: [https://github.com/fmder/pynolh/](https://github.com/fmder/pynolh/)
The python package: [https://pypi.org/project/pynolh/](https://pypi.org/project/pynolh/).

References


See Also

The main function `nolhdrDesign`. Cioppa’s NOLH design list: `NOLHdesigns`.

Examples

```r
## data(NOLHDRdesigns)
## all matrices
names(NOLHDRdesigns)
lapply(NOLHDRdesigns, tail, 2)

## The first matrix/design
NOLHDRdesigns[["nolhdr08"]]
```
A 3D orthogonal array of strength 2

Description

A 3-dimensional linear orthogonal array (OA) of strength 2 with 49 points. The design points are equally spaced into 2 dimensional coordinate planes. However by construction, such OAs satisfy a linear relation, here: \( x_1 + 3x_2 + x_3 = 0 \mod 7 \). As a consequence, the design points are contained in parallel planes orthogonal to (1,3,1). Actually, they are also contained in parallel planes orthogonal to other directions, as (2,-1,2) or (3,2,3), since the congruence relation leads to \( 2x_1 - x_2 + 2x_3 = 0 \mod 7 \) or \( 3x_1 + 2x_2 + 3x_3 = 0 \mod 7 \). For instance, they are contained in 4 parallel planes orthogonal to (2,-1,2).

Usage

```R
data(OA131)
```

Format

A data frame with 49 observations on the following 3 variables.

- `x1` first coordinate
- `x2` second coordinate
- `x3` third coordinate

Source


Examples

```R
data(OA131)

# centering and reducing to \([0,1]^3\)
OA <- (OA131 + 0.5)/7
pairs(OA, xlim=c(0,1), ylim=c(0,1))

## Not run:
library(lattice)
cloud(x3~x1+x2, data=OA, xlim=c(0,1), ylim=c(0,1), zlim=c(0,1),
      screen = list(z = 50, x = -70, y = 0))
## End(Not run)
```
Description

This design is obtained by adding a uniform noise to each coordinate of the orthogonal array OA131.

Usage

data(OA131_scrambled)

Format

A data frame with 49 observations on the following 3 variables.

- x1 first coordinate
- x2 second coordinate
- x3 third coordinate

Source


Examples

data(OA131)
data(OA131_scrambled)
pairs(OA131, xlim=c(0,1), ylim=c(0,1))
pairs(OA131_scrambled, xlim=c(0,1), ylim=c(0,1))

olhDesign

Nguyen’s Orthogonal Latin Hypercube Designs

Description

Generate the Orthogonal Latin Hypercube (OLH) designs proposed by Nguyen in 2008. These OLHs have a latin structure, an orthogonality between the main terms and the interactions (+ squares) and low correlations between the interactions (+ squares). Very larges matrices can be obtained as the number of input variables and hence the number of lines is unconstrained. When the number of input variables is a power of 2, OLHs have $d$ columns and $n = 2^d + 1$ lines (experiments). A vertical truncature is applied when the number of input variables is not a power of 2. Various normalizations can be applied.
Usage

olhDesign(dimension, range = c(0, 1))

Arguments

dimension  number of input variables
range      the scale (min and max) of the inputs. Ranges (0, 0) and (1, 1) are special cases
           and call integer ranges \((-d, d)\) and \((0, 2d)\). See the examples

Value

A list with components:

n               the number of lines/experiments
dimension      the number of columns/input variables
design         the design of experiments

Author(s)

N.K. Nguyen for the algorithm. P. Kiener for the recursive R code.

References


See Also

Cioppa's and De Rainville's NOLH designs: nolhDesign, nolhdrDesign.

Examples

## Classical normalizations
olhDesign(4, range = c(0, 0))
olhDesign(4, range = c(1, 1))
olhDesign(4, range = c(0, 1))
olhDesign(4, range = c(-1, 1))

## Change the dimnames, adjust to range (-10, 10) and round to 2 digits
xORDN(olhDesign(4), letter = "T", dgts = 2, range = c(-10, 10))

## A list of designs
lapply(1:5, function(n) olhDesign(n, range = c(-1, 1))$design)
phiP criterion

Description

Compute the $\phi_p$ criterion (strongly linked to mindist criterion)

Usage

phiP(design, p=50)

Arguments

design a matrix (or a data.frame) corresponding to the design of experiments.
p the "p" in the Lp norm which is taken

Details

The $\phi_p$ criterion is defined by the $L_p$ norm of the sum of the inverses of the design inter-point euclidean distances:

$$\phi_p = \left[ \sum_{i=1 \ldots N, i < j} d_{ij}^{-p} \right]^{\frac{1}{p}}$$

A higher value corresponds to a more regular scattering of design points.

When $p$ tends to infinity, optimizing a design with $\phi_p$ is equivalent to optimizing a design with mindist.

Value

A real number equal to the value of the $\phi_p$ criterion for the design.

Author(s)

G. Damblin & B. Iooss

References


See Also

geometric criterion (mindist)

Examples

dimension <- 2
n <- 40
X <- matrix(runif(n*dimension), n, dimension)
phiP(X)

rss2d 2D graphical tool for defect detection of Space-Filling Designs.

Description

For a 2-dimensional design, the 2D radial scanning statistic (RSS) scans angularly the domain. In each direction, it compares the distribution of projected points to their theoretical distribution under the assumption that all design points are drawn from uniform distribution. For a d-dimensional design, all pairs of dimensions are scanned. The RSS detects the defects of low discrepancy sequences or orthogonal arrays, and can be used for selecting space-filling designs.

Usage

rss2d(design, lower, upper, gof.test.type="greenwood",
gof.test.stat=NULL, transform=NULL, n.angle=360, graphics=1,
trace=TRUE, lines.lwd = 1, lines.lty = "dotted", ...)

Arguments

design a matrix or data.frame containing the d-dimensional design of experiments. The row no. i contains the values of the d input variables corresponding to simulation no. i
lower the domain lower boundaries.
upper the domain upper boundaries.
gof.test.type an optional character indicating the kind of statistical test to be used to test the goodness-of-fit of the design projections to their theoretical distribution. Several tests are available, see unif.test.statistic. Default is "greenwood".
gof.test.stat an optional number equal to the goodness-of-fit statistic at level 5%. Default is the modified test statistic for fully specified distribution (see details below).
transform an optional character indicating what type of transformation should be applied before testing uniformity. Only one choice available "spacings", that lead to over-detection. Default - and recommended - is NULL.
n.angle an optional number indicating the number of angles used. Default is 360 corresponding to a 0.5-degree discretization step. Note that the RSS curve is continuous.
graphics an optional integer indicating whether a graph should be produced. If negative, no graph is produced. If superior to 2, the RSS curve only is plotted in the worst 2D coordinate subspace (corr. to the worst value of statistic). If 1 (default), the design is also added, with its projections onto the worst (oblique) axis.

trace an optional boolean. Turn it to FALSE if you want no verbosity.

lines.lwd optional number specifying the width of the straight lines involved in the graphical outputs (axis and projections)

lines.lty optional character string specifying the type of the straight lines involved in the graphical outputs (axis and projections)

... optional graphical parameters of plot function, to draw the RSS curve.

Value

a list with components:

global.stat a matrix containing the values of the global statistic (equal to the maximum of statistic values over the RSS curve) for all pairs of dimensions.

worst.case the worst pair of dimensions, that is the one that gives the worst value of global.stat.

worst.dir the worst direction, that is the one that gives the worst value of the global statistic in the coordinate plane defined by worst.case.

stat a vector of length n.angle containing the statitic values for each angle, in the coordinate plane defined by worst.case.

angle a vector of length n.angle containing the corresponding angles used.

curve a (2*n.angle)x2 matrix containing the discretized RSS curve.

gof.test.stat the threshold at significance level 0.05 for the specified goodness-of-fit statistic. It is equal to the radius of the circle superimposed on the RSS figure.

Author(s)

O. Roustant

References


See Also

unif.test.statistic, unif.test.quantile, rss3d
Examples

```r
## Detection of defects of Sobol designs

## requires randtoolbox package
library(randtoolbox)

## In 2D
rss <- rss2d(design=sobol(n=20, dim=2), lower=c(0,0), upper=c(1,1),
             type="l", col="red")

## In 8D
## All pairs of dimensions are tried to detect the worst defect
## (according to the specified goodness-of-fit statistic).
d <- 8
n <- 10*d
rss <- rss2d(design=sobol(n=n, dim=d), lower=rep(0,d), upper=rep(1,d),
             type="l", col="red")

## Avoid this defect with scrambling?
## 1. Faure-Tezuka scrambling (type "?sobol" for more details and options)
rss <- rss2d(design=sobol(n=n, dim=d, scrambling=2), lower=rep(0,d), upper=rep(1,d),
             type="l", col="red")

## 2. Owen scrambling
rss <- rss2d(design=sobol(n=n, dim=d, scrambling=1), lower=rep(0,d), upper=rep(1,d),
             type="l", col="red")
```

---

**rss3d**

*3D graphical tool for defect detection of Space-Filling Designs.*

**Description**

For a 3-dimensional design, the 3D radial scanning statistic (RSS) scans angularly the domain. In each direction, it compares the distribution of projected points to their theoretical distribution under the assumption that all design points are drawn from uniform distribution. For a d-dimensional design, all triplets of dimensions are scanned. The RSS detects the defects of low discrepancy sequences or orthogonal arrays, and can be used for selecting space-filling designs.

**Usage**

```r
rss3d(design, lower, upper, gof.test.type = "greenwood",
gof.test.stat = NULL, transform = NULL, n.angle = 60,
       graphics = 1, trace = TRUE)
```

**Arguments**

- **design**: a matrix or data.frame containing the d-dimensional design of experiments. The row no. i contains the values of the d input variables corresponding to simulation no. i
lower the domain lower boundaries.
upper the domain upper boundaries.
gof.test.type an optional character indicating the kind of statistical test to be used to test the
    goodness-of-fit of the design projections to their theoretical distribution. Several
tests are available, see unif.test.statistic. Default is "greenwood".
gof.test.stat an optional number equal to the goodness-of-fit statistic at level 5%. Default is
    the modified test statistic for fully specified distribution (see details below).
transform an optional character indicating what type of transformation should be applied
    before testing uniformity. Only one choice available "spacings", that lead to
    over-detection. Default - and recommended - is NULL.
n.angle an optional number indicating the number of angles used. Default is 60 corre-
    sponding to a 3-degree discretization step. Note that the RSS surface is contin-
    uous.
graphics an optional integer indicating whether a graph should be produced. If negative,
    no graph is produced. Otherwise (default), the design is plotted in the worst 3D
    coordinate subspace (corr. to the worst value of statistic), with its projections
    onto the worst (oblique) axis.
trace an optional boolean. Turn it to FALSE if you want no verbosity.

Details
The RSS surface is continuous. However for computational purposes, a discretization is used. The
default discretization step is tunable with n.angle.

Value
a list with components:
global.stat an array containing the values of the global statistic (equal to the maximum of
    statistic values over the RSS surface) for all triplets of dimensions.
print.out the same as global.stat, but with a user-friendly printing.
worst.case the worst triplet of dimensions, that is the one that gives the worst value of
    global.stat.
worst.dir the worst direction, that is the one that gives the worst value of the statistic in
    the coordinate 3D subspace defined by worst.case.
stat a matrix of size n.angle*n.angle containing the statistic values for each angles
    (spherical coordinates).
angle a matrix of size n.angle*n.angle containing the corresponding angles used
    (spherical coordinates).
gof.test.stat the threshold at significance level 0.05 for the specified goodness-of-fit statistic.

Author(s)
O. Roustant
## References


## See Also

unif.test.statistic, unif.test.quantile, rss2d

## Examples

```r
## An orthogonal array in 3D
data(OA131)

## centering the design points of this 7-levels design
OA <- (OA131 + 0.5)/7

## 2D projections onto coordinate axis
pairs(OA, xlim=c(0,1), ylim=c(0,1))

## Now let us look at the 3D properties with the 3D RSS (requires the rgl package)
rss <- rss3d(OA, lower=c(0,0,0), upper=c(1,1,1))

## The worst direction detected is nearly proportional to (2,-1,2)
## (type "?OA131" for explanations about this linear orthogonal array)
print(rss$worst.dir)

## Now, scramble this design
## X <- (OA131 + matrix(runif(49*3, 49, 3))/7
## or load the design obtained this way
data(OA131_scrambled)
OA2 <- OA131_scrambled

## No feature is detected by the 2D RSS:
rss <- rss2d(OA2, lower=c(0,0,0), upper=c(1,1,1))

## 4 clusters are detected by the 3D RSS:
rss <- rss3d(OA2, lower=c(0,0,0), upper=c(1,1,1))

## Defect detection of 8D Sobol sequences
## All triplets of dimensions are tried to detect the worst defect
## (according to the specified goodness-of-fit statistic).
## requires randtoolbox library to generate the Sobol sequence
## Not run:
library(randtoolbox)
d <- 8
n <- 10*d
rss <- rss3d(design=sobol(n=n, dim=d), lower=rep(0,d), upper=rep(1,d))
## End(Not run)
```
Description

Generate a Faure sequence with \( n \) experiments in \([0,1]^d\).

Usage

runif.faure(n, dimension)

Arguments

- \( n \) the number of experiments
- \( \text{dimension} \) the number of variables (<100)

Details

A quasirandom or low discrepancy sequence, such as the Faure, Halton, Hammersley, Niederreiter or Sobol sequences, is "less random" than a pseudorandom number sequence, but more useful for such tasks as approximation of integrals in higher dimensions, and in global optimization. This is because low discrepancy sequences tend to sample space "more uniformly" than random numbers. see randtoolbox or fOptions packages for other low discrepancy sequences.

Value

runif.halton returns a list containing all the input arguments detailed before, plus the following component:

- \( \text{design} \) the design of experiments

Author(s)

J. Franco

References

Faure H. (1982), Discrepance de suites associees a un systeme de numeration (en dimension s), Acta Arith., 41, 337-351

Examples

\[
f <- \text{runif.faure}(20, 2)\\
\text{plot(f$design, xlim=c(0,1), ylim=c(0,1))}\\
\text{xORDN(f, letter="T", dgts=2, range=c(-10, 10))}
\]
scaleDesign  
*Scale a Design*

**Description**

This function scales the values of the design points to values comprised in [0,1]. The scaling can be made by the Rosenblatt transformation (uniformization by applying the empirical cumulative distribution function) or by translating the design from maximum and minimum values (given for each variable).

**Usage**

```r
scaleDesign(design, min=NULL, max=NULL, uniformize=FALSE)
```

**Arguments**

- `design` a matrix (or a data.frame) corresponding to the design of experiments to scale
- `min` the vector of minimal bounds of each design variable. If not given, the minimal value of each variable is taken
- `max` the vector of maximal bounds of each design variable. If not given, the maximal value of each variable is taken
- `uniformize` boolean: TRUE to use the Rosenblatt transformation (the min and max vectors are useless in this case). If FALSE (default value), the translation from max and min values is applied

**Value**

A list containing:

- `design` the scaled design
- `min` the vector of minimal bounds that has been used
- `max` the vector of maximal bounds that has been used
- `uniformize` the value of this boolean argument
- `InitialDesign` the starting design

**Author(s)**

B. Iooss

**Examples**

```r
d <- 2
n <- 100
x <- matrix(rnorm(d*n), ncol=d)
xscale1 <- scaleDesign(x, uniformize=FALSE)
xscale2 <- scaleDesign(x, uniformize=TRUE)
par(mfrow=c(1,2))
plot(xscale1$design) ; plot(xscale2$design)
```
straussDesign

Designs based on Strauss process

Description

Space-Filling Designs based on Strauss process

Usage

straussDesign(n, dimension, RND, alpha=0.5, repulsion=0.001, NMC=1000, constraints1D=0, repulsion1D=0.0001, seed=NULL)

Arguments

n             the number of experiments
dimension     the number of input variables
RND           a real number which represents the radius of interaction
alpha         the potential power (default, fixed at 0.5)
repulsion     the repulsion parameter in the unit cube (gamma)
NMC           the number of McMC iterations (this number must be large to converge)
constraints1D 1 to impose 1D projection constraints, 0 otherwise
repulsion1D   the repulsion parameter in 1D
seed          seed for the uniform generation of number

Details

Strauss designs are Space-Filling designs initially defined from Strauss process:

$$\pi(X) = k_\gamma s(X)$$

where $s(X)$ is the number of pairs of points $(x^i, x^j)$ of the design $X = (x^1, \ldots, x^n)$ that are separated by a distance no greater than the radius of interaction RND, $k$ is the normalizing constant and $\gamma$ is the repulsion parameter. This distribution corresponds to the particular case $\alpha=0$.

For the general case, a stochastic simulation is used to construct a Markov chain which converges to a spatial density of points $\pi(X)$ described by the Strauss-Gibbs potential. In practice, the Metropolis-Hastings algorithm is implemented to simulate a distribution of points which converges to the stationary law:

$$\pi(X) \propto exp(-U(X))$$

with a potential $U$ defined by:

$$U(X) = \beta \sum_{1 \leq i < j \leq n} \varphi(\|x^i - x^j\|)$$

where $\beta = -\ln \gamma$, $\varphi(h) = (1 - \frac{h}{RND})^\alpha$ if $h \leq RND$ and 0 otherwise.
The input parameters of `straussDesign` function can be interpreted as follows:

- `RND` is used to compute the number of pairs of points of the design separated by a distance no more than `RND`. A point is said "in interaction" with another if the spheres of radius `RND/2` centered on these points intersect.
- `alpha` is the potential power $\alpha$. The case $alpha=0$ corresponds to Strauss process (0-1 potential).
- `repulsion` is equal to the $\gamma$ parameter of the Strauss process. Note that $\gamma$ belongs to $[0,1]$.
- `constraints1D` allows to specify some constraints into the margin. If `constraints1D==1`, two repulsion parameters are needed: one for the all space (`repulsion`) and the other for the 1D projection (`repulsion1D`). Default values are `repulsion=0.001` and `repulsion1D=0.001`. Note that the value of the radius of interaction in the one-dimensional axis is not an input parameter and is automatically fixed at $0.75/n$.

Value

A list containing:

- `n` the number of experiments
- `dimension` the number $d$ of variables
- `design_init` the initial distribution of $n$ points $[0,1]^d$
- `radius` the radius of interaction
- `alpha` the potential power $\alpha$
- `repulsion` the repulsion parameter $\gamma$
- `NMC` the number of iterations McMC
- `constraints1D` an integer indicating if constraints on the factorial axis are imposed. If its value is different from zero, a component `repulsion1D` containing the value of the repulsion parameter $\gamma$ in dimension 1 is added at the list.
- `design` the design of experiments in $[0,1]^d$
- `seed` the seed corresponding to the design

Author(s)

J. Franco

References


Examples

```r
## Strauss-Gibbs designs in dimension 2 (n=20 points)
S1 <- straussDesign(n=20, dimension=2, RND=0.2)
plot(S1$design, xlim=c(0,1), ylim=c(0,1))
theta <- seq(0,2*pi, by=2*pi/(100 - 1))
for(i in 1:S1$n){
  lines(S1$design[i,1]+S1$radius/2*cos(theta),
  S1$design[i,2]+S1$radius/2*sin(theta),
```
## 2D-Strauss design

```r
S2 <- straussDesign(n=20, dimension=2, RND=0.2, NMC=200,
                           constraints1D=0, alpha=0, repulsion=0.01)
plot(S2$design, xlim=c(0,1), ylim=c(0,1))
```

## 2D-Strauss designs with constraints on the axis

```r
S3 <- straussDesign(n=20, dimension=2, RND=0.18, NMC=200,
                          constraints1D=1, alpha=0.5, repulsion=0.1, repulsion1D=0.01)
plot(S3$design, xlim=c(0,1), ylim=c(0,1))
rug(S3$design[,1], side=1)
rug(S3$design[,2], side=2)
```

## Change the dimnames, adjust to range (-10, 10) and round to 2 digits

```r
xORDN(S3, letter="T", dgts=2, range=c(-10, 10))
```

---

### unif.test.quantile

**Quantile of some uniformity tests**

**Description**

Computes the quantile of a uniformity test at a given significance level (see available tests and levels below).

**Usage**

```r
unif.test.quantile(type, n, alpha)
```

**Arguments**

- **type**: a character indicating which test is used. The choices are the following: "greenwood", "qm" (for Quesenberry-Miller), "ks" (Kolmogorov-Smirnov), "cvm" (Cramer-Von Mises) and "V" (D+ + D- from Kolmogorov-Smirnov).
- **n**: an integer equal to the sample size.
- **alpha**: a real number equal to significance level. At present stage, only four values are available: 0.1, 0.05, 0.025 and 0.01.

**Details**

Modified statistics are used. For \( \alpha = 0.05 \), the quantile is (see D Agostino and Stephens, 1986, section 4.4.): 1.358/(sqrt(n) + 0.12 + 0.11/sqrt(n)) for Kolmogorov-Smirnov and 0.461/(1+1/n) + 0.4/n - 0.6/n^2 for Cramer-von Mises. When the design size is < 20, the corrected value seems to be a good approximation, but the non asymptotical value should be preferred.
**Value**

A real number equal to the quantile of the specified test at significance level \( \alpha \) for \( n \) observations.

**Author(s)**

O. Roustant

**References**


**See Also**

`unif.test.statistic`, `rss2d`, `rss3d`

---

**Description**

Computes the statistic of a uniformity test (see available tests below).

**Usage**

`unif.test.statistic(x, type, transform=NULL)`

**Arguments**

- `x` a vector containing the sample values.
- `type` a character indicating which test is used. The choices are the following: "greenwood", "qm" (for Quesenberry-Miller), "ks" (Kolmogorov-Smirnov), "cvm" (Cramer-Von Mises) and "V" (\( D^+ + D^- \) from Kolmogorov-Smirnov).
- `transform` an optional character indicating what type of transformation should be applied before testing uniformity. Default is NULL.

**Value**

A real number equal to the statistic of the specified test.

**Author(s)**

O. Roustant

**References**

See Also

`unif.test.quantile`, `rss2d`

unscaleDesign

Unscale a Design

Description

This function unscales the values of a scaled design (values in [0,1]). The unscaling can be made by the inverse Rosenblatt transformation (by applying the empirical quantile function given by another design) or by translating the design from maximum and minimum values (given for each variable).

Usage

`unscaleDesign(design, min=NULL, max=NULL, uniformize=FALSE, InitialDesign=NULL)`

Arguments

- `design` a matrix (or a data.frame) corresponding to the design of experiments to unscale
- `min` the vector of minimal bounds of each design variable
- `max` the vector of maximal bounds of each design variable
- `uniformize` boolean: TRUE to use the inverse Rosenblatt transformation (the min and max vectors are useless in this case). If FALSE (default value), the translation from max and min values is applied
- `InitialDesign` If the inverse Rosenblatt transformation is applied (uniformize = TRUE): a matrix (or a data.frame) corresponding to the design which gives the empirical quantiles

Value

A list containing:

- `design` the unscaled design
- `min` the vector of minimal bounds that has been used
- `max` the vector of maximal bounds that has been used
- `uniformize` the value of this boolean argument

Author(s)

B. Iooss
wspDesign

Examples

\[
d <- 2
n <- 100
x <- matrix(rnorm(d*n), ncol=d)
xscale <- scaleDesign(x, uniformize=TRUE)
xunscale1 <- unscaleDesign(xscale$design, uniformize=TRUE, InitialDesign=x)
xunscale2 <- unscaleDesign(xscale$design, 
min=c(min(x[,1]), min(x[,2])), max = c(max(x[,1]), max(x[,2])))
par(mfrow=c(2,2))
plot(x) ; plot(xscale$design)
plot(xunscale1$design) ; plot(xunscale2$design)
\]

wspDesign

**WSP algorithm**

Description

The WSP (Wooton, Sergent, Phan-Tan-Luu) algorithm is an iterative algorithm based on suppression of some experiments from an initial design in each step. WSP leads to a space filling design.

Usage

wspDesign(design, dmin, init = "center")

Arguments

- **design**: a matrix (or a data.frame) corresponding to the design of experiments
- **dmin**: a minimum bound for mindist value of the final design
- **init**: defines the initialization point (input coordinates) of the algorithm: "center" (default value) takes the central point of the domain "random" takes a random point inside the domain

Details

WSP enables to create a design D which is such that mindist(D)>dmin. However, it cannot assess the number of experiments. Similarly to straussDesign function, WSP is a powerful algorithm to construct space filling designs in high dimension.

Value

A list containing:

- `InitialDesign` the starting design
- `dmin` minimum bound for mindist value of the final design
- `design` the matrix of the final design
- `ResidualDesign` the matrix of the residual design (points of InitialDesign not in design)
Author(s)
G. Damblin & B. Iooss

References

Examples
dimension <- 2
n <- 100
X <- matrix(runif(n*dimension), n, dimension)
m <- wspDesign(X, 0.1)
plot(m$design)
xDRDN(m, letter = "T", dgts = 2, range = c(-10, 10))

---

**xDRDN**

*Extract a Design and Give it a Range and Dimnames*

**Description**
Extract a design contained in a list (i.e. with a `design` item), adjust the range, give it dimnames and finally round the values to a certain number of digits. Colnames will look like (A,B,C), (X1,X2,X3), (X01,X02,X03), (X001,X002,X003).

**Usage**

```
xDRDN(obj, width = 1, letter = "X", dgts = NULL, range = NULL)
```

**Arguments**

- **obj** a list that contains a design item. Matrix or data.frame are also accepted
- **width** the digit width in colnames (to write for instance X1, X01, X001). If 0, colnames are filled with capital and small letters (without letters I and i) up to 50 columns
- **letter** the generic letter used in colnames
- **dgts** the number of digits to which the design is rounded
- **range** a vector c(min, max) to adjust the range of the design. The default NULL keeps the original range. Special ranges c(0, 0) and c(1, 1) are not accepted

**Value**
A rounded matrix or a data.frame with appropriate dimnames and an adjusted range.
Examples

`xDRDN(lhsDesign(5, 12))`
`xDRDN(lhsDesign(5, 12), width = 2, letter = "V", dgts = 2, range = c(-10, 10))`
`head(xDRDN(olhDesign(50, range = c(1,1)), width = 0, letter = "Z"), 3)`
`head(xDRDN(olhDesign(51, range = c(1,1)), width = 0, letter = "Z"), 3)`
Index

* datasets
  NOLHdesigns, 25
  NOLHDRdesigns, 27
  OA131, 29
  OA131_scrambled, 30

* design
  coverage, 4
  discrepancyCriteria, 6
  discrepESE_LHS, 7
  discrepSA_LHS, 9
  dmaxDesign, 11
  factDesign, 13
  faureprimeDesign, 14
  lhsDesign, 16
  maximinESE_LHS, 17
  maximinSA_LHS, 19
  meshRatio, 21
  mindist, 22
  mstCriteria, 23
  nolhDesign, 24
  nolhdrDesign, 26
  olhDesign, 30
  phiP, 32
  rss2d, 33
  rss3d, 35
  runif.faure, 38
  straussDesign, 40
  unif.test.quantile, 42
  unif.test.statistic, 43
  wspDesign, 45

* package
  DiceDesign-package, 2
  DiceDesign (DiceDesign-package), 2
  DiceDesign-package, 2
  discrepancyCriteria, 5, 6, 9, 10, 18, 20, 21, 23
  discrepESE_LHS, 7, 10, 17, 18, 20
  discrepSA_LHS, 9, 9, 17, 18, 20
  dmaxDesign, 11
  factDesign, 13
  faureprimeDesign, 14
  lhsDesign, 9, 10, 16, 18, 20
  maximinESE_LHS, 9, 10, 17, 17, 20
  maximinSA_LHS, 9, 10, 17, 18, 19
  meshRatio, 5, 7, 21, 21, 23
  mindist, 5, 7, 9, 10, 18, 20, 21, 22, 33
  mstCriteria, 23
  nolhDesign, 24, 25–27, 31
  NOLHdesigns, 24, 25, 25, 26, 28
  nolhdrDesign, 25, 26, 27, 28, 31
  NOLHDRdesigns, 26, 27, 27
  OA131, 29
  OA131_scrambled, 30
  olhDesign, 25, 27, 30
  phiP, 5, 7, 9, 10, 18, 20, 21, 23, 32
  rss2d, 33, 37, 43, 44
  rss3d, 34, 35, 43
  runif.faure, 14, 38
  scaleDesign, 39
  straussDesign, 40
  unif.test.quantile, 34, 37, 42, 44
  unif.test.statistic, 33, 34, 36, 37, 43, 43
  unscaleDesign, 44
  wspDesign, 45
  xDRDN, 46

48