Package 'lhs'

April 13, 2020

Title Latin Hypercube Samples

Version 1.0.2

Description Provides a number of methods for creating and augmenting Latin Hypercube Samples and Orthogonal Array Latin Hypercube Samples.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.4.0)

LinkingTo Rcpp

Imports Rcpp

Suggests testthat, DoE.base, knitr, rmarkdown, covr

URL https://github.com/bertcarnell/lhs

BugReports https://github.com/bertcarnell/lhs/issues

RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation yes

Author Rob Carnell [aut, cre]

Maintainer Rob Carnell <bertcarnell@gmail.com>

Repository CRAN

Date/Publication 2020-04-13 12:10:02 UTC

R topics documented:

`augmentLHS` *Augment a Latin Hypercube Design*

Description

Augments an existing Latin Hypercube Sample, adding points to the design, while maintaining the *latin* properties of the design.

Usage

```
augmentLHS(lhs, m = 1)
```

Arguments

`lhs`	The Latin Hypercube Design to which points are to be added. Contains an existing latin hypercube design with a number of rows equal to the points in the design (simulations) and a number of columns equal to the number of variables (parameters). The values of each cell must be between 0 and 1 and uniformly distributed
`m`	The number of additional points to add to matrix `lhs`

Details

Augments an existing Latin Hypercube Sample, adding points to the design, while maintaining the *latin* properties of the design. Augmentation is perfomed in a random manner.

The algorithm used by this function has the following steps. First, create a new matrix to hold the candidate points after the design has been re-partitioned into $(n + m)^2$ cells, where n is number of points in the original `lhs` matrix. Then randomly sweep through each column (1. . . k) in the repartitioned design to find the missing cells. For each column (variable), randomly search for an empty row, generate a random value that fits in that row, record the value in the new matrix. The new matrix can contain more filled cells than `m` unles $m = 2n$, in which case the new matrix will contain exactly `m` filled cells. Finally, keep only the first m rows of the new matrix. It is guaranteed to have m full rows in the new matrix. The deleted rows are partially full. The additional candidate points are selected randomly due to the random search for empty cells.

Value

An n by k Latin Hypercube Sample matrix with values uniformly distributed on [0,1]

Author(s)

Rob Carnell

References

Stein, M. (1987) Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics*. **29**, 143–151.

See Also

[randomLHS()], [geneticLHS()], [improvedLHS()], [maximinLHS()], and [optimumLHS()] to generate Latin Hypercube Samples. [optAugmentLHS()] and [optSeededLHS()] to modify and augment existing designs.

Examples

```
set.seed(1234)
a <- randomLHS(4,3)
b <- augmentLHS(a, 2)
```

---

createAddelKemp — *Create an orthogonal array using the Addelman-Kempthorne algorithm. The addelkemp program produces* OA(2q^2, k, q, 2)*,* k <= 2q+1*, for odd prime powers* q*.*

Description

From Owen: An orthogonal array A is a matrix of n rows, k columns with every element being one of q symbols 0,...,q-1. The array has strength t if, in every n by t submatrix, the q^t possible distinct rows, all appear the same number of times. This number is the index of the array, commonly denoted lambda. Clearly, lambda*q^t=n. The notation for such an array is OA(n,k,q,t).

Usage

```
createAddelKemp(q, ncol, bRandom = TRUE)
```

Arguments

q	the number of symbols in the array
ncol	number of parameters or columns
bRandom	should the array be randomized

Value

an orthogonal array

References

Owen, Art. Orthogonal Arrays for: Computer Experiments, Visualizations, and Integration in high dimenstions. http://lib.stat.cmu.edu/designs/oa.c. 1994 S. Addelman and O. Kempthorne (1961) Annals of Mathematical Statistics, Vol 32 pp 1167-1176.

See Also

Other methods to create orthogonal arrays [createBoseBush()], [createBose()], [createAddelKemp3()], [createAddelKempN()], [createBusht()], [createBoseBushl()]

Examples

```
A <- createAddelKemp(3, 3, TRUE)
B <- createAddelKemp(3, 5, FALSE)
```

`createAddelKemp3` *Create an orthogonal array using the Addelman-Kempthorne algorithm with* `2q^3` *rows. The addelkemp3 program produces* `OA( 2*q^3, k, q, 2 )`, `k <= 2q^2+2q+1`, *for prime powers* `q`. `q` *may be an odd prime power, or* `q` *may be 2 or 4.*

Description

From Owen: An orthogonal array `A` is a matrix of `n` rows, `k` columns with every element being one of `q` symbols `0,...,q-1`. The array has strength `t` if, in every `n` by `t` submatrix, the `q^t` possible distinct rows, all appear the same number of times. This number is the index of the array, commonly denoted `lambda`. Clearly, `lambda*q^t=n`. The notation for such an array is `OA( n,k,q,t )`.

Usage

```
createAddelKemp3(q, ncol, bRandom = TRUE)
```

Arguments

q	the number of symbols in the array
ncol	number of parameters or columns
bRandom	should the array be randomized

Value

an orthogonal array

References

Owen, Art. Orthogonal Arrays for: Computer Experiments, Visualizations, and Integration in high dimensions. http://lib.stat.cmu.edu/designs/oa.c. 1994 S. Addelman and O. Kempthorne (1961) Annals of Mathematical Statistics, Vol 32 pp 1167-1176.

See Also

Other methods to create orthogonal arrays [createBushBush()], [createBose()], [createAddelKemp()], [createAddelKempN()], [createBusht()], [createBoseBushl()]

Examples

```
A <- createAddelKemp3(3, 3, TRUE)
B <- createAddelKemp3(3, 5, FALSE)
```

---

createAddelKempN — *Create an orthogonal array using the Addelman-Kempthorne algorithm with alternate strength*

Description

Create an orthogonal array using the Addelman-Kempthorne algorithm with alternate strength

Usage

```
createAddelKempN(q, ncol, exponent, bRandom = TRUE)
```

Arguments

`q`	the number of symbols in the array
`ncol`	number of parameters or columns
`exponent`	the exponent on q
`bRandom`	should the array be randomized

Value

an orthogonal array

See Also

Other methods to create orthogonal arrays [createBoseBush()], [createBose()], [createBush()], [createAddelKemp()], [createAddelKemp3()], [createBusht()], [createBoseBushl()]

Examples

```
A <- createAddelKempN(3, 4, 3, TRUE)
B <- createAddelKempN(3, 4, 4, TRUE)
```

`createBose` *Create an orthogonal array using the Bose algorithm. The bose program produces* `OA( q^2, k, q, 2 )`, `k <= q+1` *for prime powers* `q`.

Description

From Owen: An orthogonal array `A` is a matrix of `n` rows, `k` columns with every element being one of `q` symbols `0,...,q-1`. The array has strength `t` if, in every `n` by `t` submatrix, the `q^t` possible distinct rows, all appear the same number of times. This number is the index of the array, commonly denoted `lambda`. Clearly, `lambda*q^t=n`. The notation for such an array is `OA( n,k,q,t )`.

Usage

```
createBose(q, ncol, bRandom = TRUE)
```

Arguments

`q`	the number of symbols in the array
`ncol`	number of parameters or columns
`bRandom`	should the array be randomized

Value

an orthogonal array

References

Owen, Art. Orthogonal Arrays for: Computer Experiments, Visualizations, and Integration in high dimenstions. http://lib.stat.cmu.edu/designs/oa.c. 1994 R.C. Bose (1938) Sankhya Vol 3 pp 323-338

See Also

Other methods to create orthogonal arrays [createBush()], [createBoseBush()], [createAddelKemp()], [createAddelKemp3()], [createAddelKempN()], [createBusht()], [createBoseBushl()]

Examples

```
A <- createBose(3, 3, FALSE)
B <- createBose(5, 4, TRUE)
```

`createBoseBush` *Create an orthogonal array using the Bose-Bush algorithm. The bose-bush program produces* `OA( 2q^2, k, q, 2 )`, `k <= 2q+1`, *for powers of 2,* `q=2^r`.

Description

From Owen: An orthogonal array `A` is a matrix of `n` rows, `k` columns with every element being one of `q` symbols `0,...,q-1`. The array has strength `t` if, in every `n` by `t` submatrix, the `q^t` possible distinct rows, all appear the same number of times. This number is the index of the array, commonly denoted `lambda`. Clearly, `lambda*q^t=n`. The notation for such an array is `OA( n,k,q,t )`.

Usage

```
createBoseBush(q, ncol, bRandom = TRUE)
```

Arguments

`q`	the number of symbols in the array
`ncol`	number of parameters or columns
`bRandom`	should the array be randomized

Value

an orthogonal array

References

Owen, Art. Orthogonal Arrays for: Computer Experiments, Visualizations, and Integration in high dimenstions. http://lib.stat.cmu.edu/designs/oa.c. 1994 R.C. Bose and K.A. Bush (1952) Annals of Mathematical Statistics, Vol 23 pp 508-524.

See Also

Other methods to create orthogonal arrays [createBush()], [createBose()], [createAddelKemp()], [createAddelKemp3()], [createAddelKempN()], [createBusht()], [createBoseBushl()]

Examples

```
A <- createBoseBush(4, 3, FALSE)
B <- createBoseBush(8, 3, TRUE)
```

createBoseBushl *Create an orthogonal array using the Bose-Bush algorithm with alternate strength >= 3. The bosebushl program produces* `OA( lambda*q^2, k, q, 2 )`, `k <= lambda*q+1`, *for prime powers* `q` *and* `lambda > 1`. *Both* `q` *and* `lambda` *must be powers of the same prime.*

Description

From Owen: An orthogonal array `A` is a matrix of `n` rows, `k` columns with every element being one of `q` symbols `0,...,q-1`. The array has strength `t` if, in every `n` by `t` submatrix, the `q^t` possible distinct rows, all appear the same number of times. This number is the index of the array, commonly denoted `lambda`. Clearly, `lambda*q^t=n`. The notation for such an array is `OA( n,k,q,t )`.

Usage

```
createBoseBushl(q, ncol, lambda, bRandom = TRUE)
```

Arguments

`q`	the number of symbols in the array
`ncol`	number of parameters or columns
`lambda`	the lambda of the BoseBush algorithm
`bRandom`	should the array be randomized

Value

an orthogonal array

References

Owen, Art. Orthogonal Arrays for: Computer Experiments, Visualizations, and Integration in high dimenstions. http://lib.stat.cmu.edu/designs/oa.c. 1994 R.C. Bose and K.A. Bush (1952) Annals of Mathematical Statistics, Vol 23 pp 508-524.

See Also

Other methods to create orthogonal arrays [createBoseBush()], [createBose()], [createBush()], [createAddelKemp()], [createAddelKemp3()], [createAddelKempN()], [createBusht()]

Examples

```
A <- createBoseBushl(3, 3, 3, TRUE)
B <- createBoseBushl(4, 4, 16, TRUE)
```

createBush — *Create an orthogonal array using the Bush algorithm. The bush program produces* `OA( q^3, k, q, 3 )`, `k <= q+1` *for prime powers* `q`.

Description

From Owen: An orthogonal array `A` is a matrix of `n` rows, `k` columns with every element being one of `q` symbols `0,...,q-1`. The array has strength `t` if, in every `n` by `t` submatrix, the `q^t` possible distinct rows, all appear the same number of times. This number is the index of the array, commonly denoted `lambda`. Clearly, `lambda*q^t=n`. The notation for such an array is `OA( n,k,q,t )`.

Usage

```
createBush(q, ncol, bRandom = TRUE)
```

Arguments

`q`	the number of symbols in the array
`ncol`	number of parameters or columns
`bRandom`	should the array be randomized

Value

an orthogonal array

References

Owen, Art. Orthogonal Arrays for: Computer Experiments, Visualizations, and Integration in high dimenstions. http://lib.stat.cmu.edu/designs/oa.c. 1994 K.A. Bush (1952) Annals of Mathematical Statistics, Vol 23 pp 426-434

See Also

Other methods to create orthogonal arrays [createBoseBush()], [createBose()], [createAddelKemp()], [createAddelKemp3()], [createAddelKempN()], [createBusht()], [createBoseBushl()]

Examples

```
A <- createBush(3, 3, FALSE)
B <- createBush(4, 5, TRUE)
```

createBusht — *Create an orthogonal array using the Bush algorithm with alternate strength. The bush program produces* `OA( q^t, k, q, t )`, `k <= q+1`, `t>=3`, *for prime powers* `q`.

Description

From Owen: An orthogonal array `A` is a matrix of `n` rows, `k` columns with every element being one of `q` symbols `0,...,q-1`. The array has strength `t` if, in every `n` by `t` submatrix, the `q^t` possible distinct rows, all appear the same number of times. This number is the index of the array, commonly denoted `lambda`. Clearly, `lambda*q^t=n`. The notation for such an array is `OA( n,k,q,t )`.

Usage

```
createBusht(q, ncol, strength, bRandom = TRUE)
```

Arguments

`q`	the number of symbols in the array
`ncol`	number of parameters or columns
`strength`	the strength of the array to be created
`bRandom`	should the array be randomized

Value

an orthogonal array

References

Owen, Art. Orthogonal Arrays for: Computer Experiments, Visualizations, and Integration in high dimensions. http://lib.stat.cmu.edu/designs/oa.c. 1994 K.A. Bush (1952) Annals of Mathematical Statistics, Vol 23 pp 426-434

See Also

Other methods to create orthogonal arrays [createBoseBush()], [createBose()], [createAddelKemp()], [createAddelKemp3()], [createAddelKempN()], [createBoseBushl()]

Examples

```
set.seed(1234)
A <- createBusht(3, 4, 2, TRUE)
B <- createBusht(3, 4, 3, FALSE)
G <- createBusht(3, 4, 3, TRUE)
```

create_oalhs — *Create an orthogonal array Latin hypercube*

Description

Create an orthogonal array Latin hypercube

Usage

```
create_oalhs(n, k, bChooseLargerDesign, bverbose)
```

Arguments

`n`	the number of samples or rows in the LHS (integer)
`k`	the number of parameters or columns in the LHS (integer)
`bChooseLargerDesign`	should a larger oa design be chosen than the n and k requested?
`bverbose`	should information be printed with execution

Value

a numeric matrix which is an orthogonal array Latin hypercube sample

Examples

```
set.seed(34)
A <- create_oalhs(9, 4, TRUE, FALSE)
B <- create_oalhs(9, 4, TRUE, FALSE)
```

geneticLHS — *Latin Hypercube Sampling with a Genetic Algorithm*

Description

Draws a Latin Hypercube Sample from a set of uniform distributions for use in creating a Latin Hypercube Design. This function attempts to optimize the sample with respect to the S optimality criterion through a genetic type algorithm.

Usage

```
geneticLHS(
  n = 10,
  k = 2,
  pop = 100,
  gen = 4,
  pMut = 0.1,
  criterium = "S",
  verbose = FALSE
)
```

Arguments

n	The number of partitions (simulations or design points or rows)
k	The number of replications (variables or columns)
pop	The number of designs in the initial population
gen	The number of generations over which the algorithm is applied
pMut	The probability with which a mutation occurs in a column of the progeny
criterium	The optimality criterium of the algorithm. Default is S. Maximin is also supported
verbose	Print informational messages. Default is FALSE

Details

Latin hypercube sampling (LHS) was developed to generate a distribution of collections of parameter values from a multidimensional distribution. A square grid containing possible sample points is a Latin square iff there is only one sample in each row and each column. A Latin hypercube is the generalisation of this concept to an arbitrary number of dimensions. When sampling a function of k variables, the range of each variable is divided into n equally probable intervals. n sample points are then drawn such that a Latin Hypercube is created. Latin Hypercube sampling generates more efficient estimates of desired parameters than simple Monte Carlo sampling.

This program generates a Latin Hypercube Sample by creating random permutations of the first n integers in each of k columns and then transforming those integers into n sections of a standard uniform distribution. Random values are then sampled from within each of the n sections. Once the sample is generated, the uniform sample from a column can be transformed to any distribution by using the quantile functions, e.g. qnorm(). Different columns can have different distributions.

S-optimality seeks to maximize the mean distance from each design point to all the other points in the design, so the points are as spread out as possible.

Genetic Algorithm:

1. Generate pop random latin hypercube designs of size n by k
2. Calculate the S optimality measure of each design
3. Keep the best design in the first position and throw away half of the rest of the population
4. Take a random column out of the best matrix and place it in a random column of each of the other matricies, and take a random column out of each of the other matricies and put it in copies of the best matrix thereby causing the progeny

5. For each of the progeny, cause a genetic mutation pMut percent of the time. The mutation is accomplished by swtching two elements in a column

Value

An n by k Latin Hypercube Sample matrix with values uniformly distributed on [0,1]

Author(s)

Rob Carnell

References

Stocki, R. (2005) A method to improve design reliability using optimal Latin hypercube sampling *Computer Assisted Mechanics and Engineering Sciences* **12**, 87–105.

Stein, M. (1987) Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics*. **29**, 143–151.

See Also

[randomLHS()], [improvedLHS()], [maximinLHS()], and [optimumLHS()] to generate Latin Hypercube Samples. [optAugmentLHS()] [optSeededLHS()], and [augtmentLHS()] to modify and augment existing designs.

Examples

```
set.seed(1234)
A <- geneticLHS(4, 3, 50, 5, .25)
```

---

improvedLHS — *Improved Latin Hypercube Sample*

Description

Draws a Latin Hypercube Sample from a set of uniform distributions for use in creating a Latin Hypercube Design. This function attempts to optimize the sample with respect to an optimum euclidean distance between design points.

Usage

```
improvedLHS(n, k, dup = 1)
```

Arguments

n	The number of partitions (simulations or design points or rows)
k	The number of replications (variables or columns)
dup	A factor that determines the number of candidate points used in the search. A multiple of the number of remaining points than can be added.

Details

Latin hypercube sampling (LHS) was developed to generate a distribution of collections of parameter values from a multidimensional distribution. A square grid containing possible sample points is a Latin square iff there is only one sample in each row and each column. A Latin hypercube is the generalisation of this concept to an arbitrary number of dimensions. When sampling a function of `k` variables, the range of each variable is divided into `n` equally probable intervals. `n` sample points are then drawn such that a Latin Hypercube is created. Latin Hypercube sampling generates more efficient estimates of desired parameters than simple Monte Carlo sampling.

This program generates a Latin Hypercube Sample by creating random permutations of the first `n` integers in each of `k` columns and then transforming those integers into `n` sections of a standard uniform distribution. Random values are then sampled from within each of the `n` sections. Once the sample is generated, the uniform sample from a column can be transformed to any distribution byusing the quantile functions, e.g. qnorm(). Different columns can have different distributions.

This function attempts to optimize the sample with respect to an optimum euclidean distance between design points.

$$Optimum distance = fracnn^{\frac{1.0}{k}}$$

Value

An `n` by `k` Latin Hypercube Sample matrix with values uniformly distributed on [0,1]

References

Beachkofski, B., Grandhi, R. (2002) Improved Distributed Hypercube Sampling *American Institute of Aeronautics and Astronautics Paper* **1274**.

This function is based on the MATLAB program written by John Burkardt and modified 16 Feb 2005 `http://www.csit.fsu.edu/~burkardt/m_src/ihs/ihs.m`

See Also

[randomLHS()], [geneticLHS()], [maximinLHS()], and [optimumLHS()] to generate Latin Hypercube Samples. [optAugmentLHS()], [optSeededLHS()], and [augmentLHS()] to modify and augment existing designs.

Examples

```
set.seed(1234)
A <- improvedLHS(4, 3, 2)
```

`maximinLHS` *Maximin Latin Hypercube Sample*

Description

Draws a Latin Hypercube Sample from a set of uniform distributions for use in creating a Latin Hypercube Design. This function attempts to optimize the sample by maximizing the minium distance between design points (maximin criteria).

Usage

```
maximinLHS(
  n,
  k,
  method = "build",
  dup = 1,
  eps = 0.05,
  maxIter = 100,
  optimize.on = "grid",
  debug = FALSE
)
```

Arguments

`n`	The number of partitions (simulations or design points or rows)
`k`	The number of replications (variables or columns)
`method`	`build` or `iterative` is the method of LHS creation. `build` finds the next best point while constructing the LHS. `iterative` optimizes the resulting sample on [0,1] or sample grid on [1,N]
`dup`	A factor that determines the number of candidate points used in the search. A multiple of the number of remaining points than can be added. This is used when `method="build"`
`eps`	The minimum percent change in the minimum distance used in the `iterative` method
`maxIter`	The maximum number of iterations to use in the `iterative` method
`optimize.on`	`grid` or `result` gives the basis of the optimization. `grid` optimizes the LHS on the underlying integer grid. `result` optimizes the resulting sample on [0,1]
`debug`	prints additional information about the process of the optimization

Details

Latin hypercube sampling (LHS) was developed to generate a distribution of collections of parameter values from a multidimensional distribution. A square grid containing possible sample points is a Latin square iff there is only one sample in each row and each column. A Latin hypercube is the generalisation of this concept to an arbitrary number of dimensions. When sampling a function of `k` variables, the range of each variable is divided into `n` equally probable intervals. `n` sample points are then drawn such that a Latin Hypercube is created. Latin Hypercube sampling generates more efficient estimates of desired parameters than simple Monte Carlo sampling.

This program generates a Latin Hypercube Sample by creating random permutations of the first `n` integers in each of `k` columns and then transforming those integers into `n` sections of a standard uniform distribution. Random values are then sampled from within each of the n sections. Once the sample is generated, the uniform sample from a column can be transformed to any distribution by using the quantile functions, e.g. qnorm(). Different columns can have different distributions.

Here, values are added to the design one by one such that the maximin criteria is satisfied.

Value

An n by k Latin Hypercube Sample matrix with values uniformly distributed on [0,1]

References

Stein, M. (1987) Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics*. **29**, 143–151.

This function is motivated by the MATLAB program written by John Burkardt and modified 16 Feb 2005 http://www.csit.fsu.edu/~burkardt/m_src/ihs/ihs.m

See Also

[randomLHS()], [geneticLHS()], [improvedLHS()] and [optimumLHS()] to generate Latin Hypercube Samples. [optAugmentLHS()], [optSeededLHS()], and [augmentLHS()] to modify and augment existing designs.

Examples

```
set.seed(1234)
A1 <- maximinLHS(4, 3, dup=2)
A2 <- maximinLHS(4, 3, method="build", dup=2)
A3 <- maximinLHS(4, 3, method="iterative", eps=0.05, maxIter=100, optimize.on="grid")
A4 <- maximinLHS(4, 3, method="iterative", eps=0.05, maxIter=100, optimize.on="result")
```

---

oa_to_oalhs *Create a Latin hypercube from an orthogonal array*

Description

Create a Latin hypercube from an orthogonal array

Usage

```
oa_to_oalhs(n, k, oa)
```

Arguments

n	the number of samples or rows in the LHS (integer)
k	the number of parameters or columns in the LHS (integer)
oa	the orthogonal array to be used as the basis for the LHS (matrix of integers) or data.frame of factors

Value

a numeric matrix which is a Latin hypercube sample

Examples

```
oa <- createBose(3, 4, TRUE)
B <- oa_to_oalhs(9, 4, oa)
```

`optAugmentLHS` *Optimal Augmented Latin Hypercube Sample*

Description

Augments an existing Latin Hypercube Sample, adding points to the design, while maintaining the *latin* properties of the design. This function attempts to add the points to the design in an optimal way.

Usage

```
optAugmentLHS(lhs, m = 1, mult = 2)
```

Arguments

`lhs`	The Latin Hypercube Design to which points are to be added
`m`	The number of additional points to add to matrix `lhs`
`mult`	`m*mult` random candidate points will be created.

Details

Augments an existing Latin Hypercube Sample, adding points to the design, while maintaining the *latin* properties of the design. This function attempts to add the points to the design in a way that maximizes S optimality.

S-optimality seeks to maximize the mean distance from each design point to all the other points in the design, so the points are as spread out as possible.

Value

An `n` by `k` Latin Hypercube Sample matrix with values uniformly distributed on [0,1]

References

Stein, M. (1987) Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics*. **29**, 143–151.

See Also

[randomLHS()], [geneticLHS()], [improvedLHS()], [maximinLHS()], and [optimumLHS()] to generate Latin Hypercube Samples. [optSeededLHS()] and [augmentLHS()] to modify and augment existing designs.

Examples

```
set.seed(1234)
a <- randomLHS(4,3)
b <- optAugmentLHS(a, 2, 3)
```

---

optimumLHS *Optimum Latin Hypercube Sample*

Description

Draws a Latin Hypercube Sample from a set of uniform distributions for use in creating a Latin Hypercube Design. This function uses the Columnwise Pairwise (CP) algorithm to generate an optimal design with respect to the S optimality criterion.

Usage

```
optimumLHS(n = 10, k = 2, maxSweeps = 2, eps = 0.1, verbose = FALSE)
```

Arguments

n	The number of partitions (simulations or design points or rows)
k	The number of replications (variables or columns)
maxSweeps	The maximum number of times the CP algorithm is applied to all the columns.
eps	The optimal stopping criterion. Algorithm stops when the change in optimality measure is less than eps*100% of the previous value.
verbose	Print informational messages

Details

Latin hypercube sampling (LHS) was developed to generate a distribution of collections of parameter values from a multidimensional distribution. A square grid containing possible sample points is a Latin square iff there is only one sample in each row and each column. A Latin hypercube is the generalisation of this concept to an arbitrary number of dimensions. When sampling a function of k variables, the range of each variable is divided into n equally probable intervals. n sample points are then drawn such that a Latin Hypercube is created. Latin Hypercube sampling generates more efficient estimates of desired parameters than simple Monte Carlo sampling.

This program generates a Latin Hypercube Sample by creating random permutations of the first n integers in each of k columns and then transforming those integers into n sections of a standard uniform distribution. Random values are then sampled from within each of the n sections. Once the sample is generated, the uniform sample from a column can be transformed to any distribution by using the quantile functions, e.g. qnorm(). Different columns can have different distributions.

S-optimality seeks to maximize the mean distance from each design point to all the other points in the design, so the points are as spread out as possible.

This function uses the CP algorithm to generate an optimal design with respect to the S optimality criterion.

Value

An n by k Latin Hypercube Sample matrix with values uniformly distributed on [0,1]

References

Stocki, R. (2005) A method to improve design reliability using optimal Latin hypercube sampling *Computer Assisted Mechanics and Engineering Sciences* **12**, 87–105.

See Also

[randomLHS()], [geneticLHS()], [improvedLHS()] and [maximinLHS()] to generate Latin Hypercube Samples. [optAugmentLHS()], [optSeededLHS()], and [augmentLHS()] to modify and augment existing designs.

Examples

```
A <- optimumLHS(4, 3, 5, .05)
```

---

optSeededLHS — *Optimum Seeded Latin Hypercube Sample*

Description

Augments an existing Latin Hypercube Sample, adding points to the design, while maintaining the *latin* properties of the design. This function then uses the columnwise pairwise (CP) algoritm to optimize the design. The original design is not necessarily maintained.

Usage

```
optSeededLHS(seed, m = 0, maxSweeps = 2, eps = 0.1, verbose = FALSE)
```

Arguments

seed	The number of partitions (simulations or design points)
m	The number of additional points to add to the seed matrix seed. default value is zero. If m is zero then the seed design is optimized.
maxSweeps	The maximum number of times the CP algorithm is applied to all the columns.
eps	The optimal stopping criterion
verbose	Print informational messages

Details

Augments an existing Latin Hypercube Sample, adding points to the design, while maintaining the *latin* properties of the design. This function then uses the CP algoritm to optimize the design. The original design is not necessarily maintained.

Value

An n by k Latin Hypercube Sample matrix with values uniformly distributed on [0,1]

References

Stein, M. (1987) Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics*. **29**, 143–151.

See Also

[randomLHS()], [geneticLHS()], [improvedLHS()], [maximinLHS()], and [optimumLHS()] to generate Latin Hypercube Samples. [optAugmentLHS()] and [augmentLHS()] to modify and augment existing designs.

Examples

```
set.seed(1234)
a <- randomLHS(4,3)
b <- optSeededLHS(a, 2, 2, .1)
```

---

randomLHS *Construct a random Latin hypercube design*

Description

randomLHS(4,3) returns a 4x3 matrix with each column constructed as follows: A random permutation of (1,2,3,4) is generated, say (3,1,2,4) for each of K columns. Then a uniform random number is picked from each indicated quartile. In this example a random number between .5 and .75 is chosen, then one between 0 and .25, then one between .25 and .5, finally one between .75 and 1.

Usage

```
randomLHS(n, k, preserveDraw = FALSE)
```

Arguments

n	the number of rows or samples
k	the number of columns or parameters/variables
preserveDraw	should the draw be constructed so that it is the same for variable numbers of columns?

Value

a Latin hypercube sample

Examples

```
a <- randomLHS(5, 3)
```

runifint *Create a Random Sample of Uniform Integers*

Description

Create a Random Sample of Uniform Integers

Usage

```
runifint(n = 1, min_int = 0, max_int = 1)
```

Arguments

n	The number of samples
min_int	the minimum integer x >= min_int
max_int	the maximum integer x <= max_int

Value

the sample sample of size n

Index