Package ‘blocksdesign’

January 11, 2019

Type Package

Title Nested and Crossed Block Designs for Factorial, Fractional
Factorial and Unstructured Treatment Sets

Version 3.3

Date 2019-01-14

Author R. N. Edmondson.

Maintainer Rodney Edmondson <rodney.edmondson@gmail.com>

Depends R (>= 3.1.0)

Description Constructs D-optimal or near D-optimal nested and crossed
block designs for unstructured or general factorial treatment designs.
The treatment design, if required, is found from a defined model
design formula. The block design is found from a defined
set of block factors and is conditional on the defined treatment design.
The block factors are added in sequence and each added block factor
is optimized conditional on all previously added block factors.
The block design can have repeated nesting down to any required
depth of nesting with either simple nested blocks or a
crossed blocks design at each level of nesting. Outputs include a table
showing the allocation of treatments to blocks and tables showing
the achieved D-efficiency factors for each block and treatment design.

License GPL (>= 2)

Imports lme4, plyr, crossdes

LazyData true

RoxygenNote 6.1.1

Encoding UTF-8

Suggests knitr, rmarkdown, R.rsp

VignetteBuilder knitr,R.rsp

NeedsCompilation no

Repository CRAN

Date/Publication 2019-01-11 17:00:13 UTC
blocksdesign-package

R topics documented:

blocksdesign-package .................................................. 2
A_bound ................................................................. 3
blockEfficiencies ......................................................... 4
blocks ................................................................. 5
design ............................................................... 7
durban ............................................................... 12
EstEffics ............................................................ 12
fullModel ............................................................ 13
HCF ................................................................. 14
isPrime ............................................................. 15
orthogLS ............................................................ 15

Index 17

blocksdesign-package  Blocks design package

Description

The blocksdesign package provides functionality for the construction of nested or crossed block
designs for general linear model treatment designs.

Details

Block designs group experimental units into homogeneous blocks to provide maximum precision
of estimation of treatment effects within blocks. The most basic type of block design is a complete
randomized blocks design where each block contains one or more complete replicate sets of treat-
ments. Complete randomized block designs estimate all treatment effects fully within individual
blocks and are usually the best choice for small experiments. However, for large experiments, the
variability within complete blocks can be large and then it may be beneficial to sub-divide each
complete block into smaller more homogeneous incomplete blocks.

Block designs with a single level of nesting are widely used in practical research but sometimes
for very large experiments a single set of nested blocks may still be too large to give good control
of intra-block variability. In this situation, a second set of incomplete blocks can be nested within
the first set to reduce the intra-block variability still further. This process of recursive nesting can
be repeated as often as required until the bottom set of blocks is sufficiently small to give adequate
control of intra-block variability.

Sometimes it can be advantageous to use a double blocking system in which one set of blocks, usu-
ally called row blocks, is crossed with a second set of blocks, usually called column blocks. Double
blocking systems can be valuable for controlling block effects in two dimensions simultaneously.

The blocksdesign package provides functionality for the construction of general multi-level block
designs with nested or crossed blocks for any feasible depth of nesting. The design algorithm
proceeds recursively with each nested set of blocks optimized conditionally within the levels of each
preceeding set of blocks. The analysis of incomplete block designs is complex but the availability of
modern computers and modern software, for example the R mixed model software package lme4
(Bates et. al. 2014), makes the analysis of any feasible nested block designs with any depth of nesting practicable.

The blocksdesign package has two design functions:

i) **blocks**: This is a simple recursive function for nested block designs for unstructured treatment sets. The function generates designs for treatments with arbitrary levels of replication and with arbitrary depth of nesting where each successive set of blocks is optimized within the levels of each preceding set of blocks using conditional D-optimality. Special block designs such as lattice designs or latin or Trojan square designs are constructed algebraically. The outputs from the blocks function include a data frame showing the allocation of treatments to blocks for each plot of the design and a table showing the achieved D- and A-efficiency factors for each set of nested blocks together with A-efficiency upper bounds, where available. A plan showing the allocation of treatments to blocks in the bottom level of the design is also included in the output.

i) **design**: This is a general purpose function for unstructured or general qualitative or quantitative factorial treatment sets. The function first finds a D-optimal or near D-optimal treatment design of the required size, possibly a simple unstructured treatment set. The function then finds a D-optimal or near D-optimal block design for that treatment design based on a set of defined block factors, if present. The blocks design algorithm builds the blocks design by sequentially adding block factors where each block factor is optimized conditional on all previous block factors. Sequential optimization allows the blocking factors to be fitted in order of importance with the largest and most important blocks fitted first and the smaller and less important blocks fitted subsequently. If there are no defined block factors, the algorithm assumes a completely randomised treatment design. The outputs include a data frame of the block and treatment factors for each plot and a table showing the achieved D-efficiency factors for each set of nested or crossed blocks. Fractional factorial efficiency factors based on the generalized variance of the complete factorial design are also shown (see the design documentation for more details)

**References**


---

**A_bound**

*Efficiency bounds*

**Description**

Finds upper A-efficiency bounds for regular block designs.

**Usage**

A_bound(n, v, b)

**Arguments**

- **n**: the total number of plots in the design.
- **v**: the total number of treatments in the design.
- **b**: the total number of blocks in the design.
Details

Upper bounds for the A-efficiency factors of regular block designs (see Chapter 2.8 of John and Williams 1995). Non-trivial A-efficiency upper bounds are calculated for regular block designs with equal block sizes and equal replication. All other designs return NA.

References


Examples

# 50 plots, 10 treatments and 10 blocks for a design with 5 replicates and blocks of size 5
A_bound(n=50, v=10, b=10)

blockEfficiencies   D-Efficiency factors

Description

Finds D-efficiency for general treatment and block designs.

Usage

blockEfficiencies(TF, BF, treatments_model)

Arguments

TF   the treatments factor data frame
BF   the block factors data frame
treatments_model
      a model formula for the required treatments design where the default formula assumes a fully crossed factorial treatment model.

Details

efficiency factors of regular block designs
**blocks**

---

**Block designs for unstructured treatment sets**

---

**Description**

Constructs randomized nested block designs for unstructured treatment sets with any feasible depth of nesting.

**Usage**

```r
blocks(treatments, replicates, blocks = NULL, searches = NULL,
       seed = NULL, jumps = 1)
```

**Arguments**

- `treatments` a partition of the total required number of treatments into equally replicated treatment sets, possibly a complete partition into individual treatments.
- `replicates` a set of treatment replication numbers with one replication number for each partitioned treatment set, possibly a complete set of treatment replication numbers.
- `blocks` the number of blocks nested in each preceding block for each level of nesting from the top-level block downwards.
- `searches` the maximum number of local optima searched for a design optimization. The default number decreases as the design size increases.
- `seed` an integer initializing the random number generator. The default is a random seed.
- `jumps` the number of pairwise random treatment swaps used to escape a local maxima. The default is a single swap.

**Details**

Constructs randomized nested block designs with arbitrary depth of nesting for arbitrary unstructured treatment sets.

The `treatments` parameter is a set of numbers that partitions the total number of treatments into equally replicated treatment sets while the `replicates` parameter is a matching set of numbers that defines the replication of each equally replicated treatment set.

The `blocks` parameter, if any, defines the number of blocks for each level of nesting from the highest to the lowest. The first number, if any, is the number of nested row blocks in the first-level of nesting, the second number, if any, is the number of nested row blocks in the second-level of nesting and so on down to any required feasible depth of nesting.

Block sizes are as nearly equal as possible and will never differ by more than a single plot for any particular block classification.

Unreplicated treatments are allowed and any simple nested block design can be augmented by any number of single unreplicated treatments to give augmented blocks that never differ in size by more than a single plot. However, it may sometimes be preferable to find an efficient block design for the replicated treatments and then add the unreplicated treatments to the design heuristically.
Square lattice designs are resolvable incomplete block designs for \( r \) replicates of \( p^2 \) treatments arranged in blocks of size \( p \) where \( r < p+2 \) for prime or prime power \( p \) or \( r < 4 \) for general \( p \). Square lattice designs are constructed algebraically from Latin squares or MOLS.

Rectangular lattice designs are resolvable incomplete block designs for \( r \) replicates of \( (p-1)^2 \) treatments arranged in blocks of size \( p-1 \) where \( r < p+1 \) for prime or prime power \( p \). Rectangular lattice designs are constructed algebraically from Latin squares or MOLS.

Designs based on prime-power MOLS require the \texttt{MOLS} package.

All other designs are constructed numerically by optimizing a D-optimality criterion.

Outputs:

- A data frame showing the allocation of treatments to blocks with successive nested strata arranged in standard block order.

- A table showing the replication number of each treatment in the design.

- A table showing the block levels and the achieved D-efficiency and A-efficiency factor for each nested level together with A-efficiency upper bounds, where available.

- A plan showing the allocation of treatments to blocks in the bottom level of the design.

Value

- \texttt{Treatments}\hspace{1cm}A table showing the replication number of each treatment in the design.
- \texttt{Design}\hspace{1cm}Data frame giving the optimized block and treatment design in plot order.
- \texttt{Plan}\hspace{1cm}Data frame showing a plan view of the treatment design in the bottom level of the design.
- \texttt{blocks_model}\hspace{1cm}The D-efficiencies and the A-efficiencies of the blocks in each nested level of the design together with A-efficiency upper-bounds, where available.
- \texttt{seed}\hspace{1cm}Numerical seed used for random number generator.
- \texttt{searches}\hspace{1cm}Maximum number of searches used for each level.
- \texttt{jumps}\hspace{1cm}Number of random treatment swaps used to escape a local maxima.

References


Examples

```r
# The number of searches in the following examples have been limited for fast execution.
# In practice, the number of searches may need to be increased for optimum results.
# Designs should be rebuilt several times to check that a near-optimum design has been found.
```
# 12 treatments x 4 replicates in 4 complete blocks with 4 sub-blocks of size 3
# rectangular lattice see Plan 10.10 Cochran and Cox 1957.
blocks(treatments=12,replicates=4,blocks=c(4,4))

# 3 treatments x 2 replicates + 2 treatments x 4 replicates in two complete randomized blocks
blocks(treatments=c(3,2),replicates=c(2,4),searches=10)

# 50 treatments x 4 replicates with 4 main blocks and 5 nested sub-blocks in each main block
blocks(treatments=50,replicates=4,blocks=c(4,5))

# as above but with 20 additional single replicate treatments, one single treatment per sub-block
blocks(treatments=c(50,20),replicates=c(4,1),blocks=c(4,5))

# 6 replicates of 6 treatments in 4 blocks of size 9 (non-binary block design)
blocks(treatments=6,replicates=6,blocks=4)

# 128 treatments x 2 replicates with two main blocks and 3 levels of nesting
blocks(128,2,c(2,2,2,2))

#' # 64 treatments x 4 replicates with 4 main blocks nested blocks of size 8 (lattice square)
blocks(64,4,c(4,8))

# 100 treatments x 4 replicates with 4 main blocks nested blocks of size 10 (lattice square)
blocks(100,4,c(4,10))

---

**design**

*General block and treatment designs.*

**Description**

Constructs D-optimal block and treatment designs for feasible combinations of nested or crossed block factors and feasible linear treatment models.

**Usage**

```r
design(treatments, blocks = NULL, treatments_model = NULL, weighting = 0.5, searches = NULL, seed = NULL, jumps = 1, fullset = TRUE)
```

**Arguments**

- `treatments` a single treatment factor or a data frame containing one or more qualitative or quantitative (numeric) level treatment factors.
- `blocks` a single block factor or a data frame containing one or more qualitative level block factors in the required order of fitting.
treatments_model

- a model formula for the required treatments design where the default formula assumes a fully crossed factorial treatment model.

weighting

- a weighting factor between 0 and 1 for weighting the interaction effects of crossed blocks factors where the default weighting is 0.5.

searches

- the maximum number of local optima searched at each stage of an optimization. The default depends on the design size.

seed

- an integer initializing the random number generator. The null default gives an arbitrary random initialization.

jumps

- the number of pairwise random treatment swaps used to escape a local maxima. The default is a single swap.

fullset

- logical parameter for the method of selection when the treatment candidate set size and the design size are equal. FALSE allows selection-with-replacement whereas TRUE requires selection of the full candidate set. Default is TRUE.

Details

The treatments object is a factor or a data frame containing one or more qualitative level treatment factors. The treatments object provides the candidate set for the treatment design.

The blocks object is a factor or a data frame containing one or more qualitative level block factors where the length of the factors defines the overall size of the required design. The blocks object must be defined even when it is just a single factor with a single factor level.

The treatment design is selected from the candidate set by choosing a set of treatments that optimizes the information matrix of the treatments design. Treatments are selected with replacement unless the size of the candidate set exactly equals the size of the required design in which case the default action is to select the full candidate set for the treatment design. This action allows any arbitrary treatment set with any arbitrary treatment replication to be input as the treatment design. If selection with replacement is required when the candidates set size exactly equals the design size, the fullset parameter should be set to FALSE.

The design criterion is the ratio of the generalized variance of the full treatment candidate set relative to the generalized variance of the optimized treatment set for the required treatment design (D-optimality). If the required design is a fractional factorial and the candidate set is a full factorial, the candidate set will be orthogonal and any design selected from the candidate set will have a relative efficiency less than or equal to 1. For a quantitative level treatment model, however, a full factorial design may not provide an optimal design and then the relative efficiency of the optimized design may well exceed 1. The efficiency factor can be used to compare different optimizations of same design with the best design having the largest efficiency.

For unstructured treatment designs, the A-efficiency factor is also shown together with an estimated A-efficiency upper-bound, where available.

The design algorithm fits the blocks design by sequentially adding block design factors in the column order of the blocks data frame. Each block factor is optimized conditional on all preceding block factors remaining fixed but ignoring all succeeding block factors. This method allows the blocking factors to be fitted in order of importance with the largest and most important blocks fitted first and the smaller and less important blocks fitted subsequently.

For crossed blocks designs, a differential weighting factor w is used to determine the relative importance of the blocks main effects versus the blocks interaction effects. If w = 0 the algorithm
fits a simple additive main effects design whereas if \( w = 1 \) the algorithm fits a fully crossed blocks design. For intermediate \( 0 < w < 1 \), the block factor interaction effects are downweighted relative to the main effects where the smaller the value of \( w \), the greater the downweighting. The default weighting is 0.5 and provided that all block effects are estimable, this weighting will give a design where main block effects are assumed to be of greater importance than block interaction effects.

For example, a design for 4 replicates of 12 treatments arranged in 4 main rows and 4 main columns with 3 plots nested within each intersection block (see examples) is known to have an optimal Trojan solution with orthogonal main rows, orthogonal main columns and rows-by-columns intersection blocks with A-efficiency = 22/31. The default weighting of 0.5 will find an optimal Trojan design whereas a weighting of \( w = 0 \) will find an optimal main column blocks design but with a sub-optimal sub-blocks design while a weighting of \( w = 1 \) will find an optimal sub-blocks design but with with a sub-optimal main columns design.

Outputs:

The principle design outputs comprise:

- A data frame showing the replication of each individual treatment taken in a standard order.
- A data frame showing the randomized allocation of treatments to blocks.
- A table showing the fractional size of the treatment design and the D-efficiency factor of that fraction.
- A table showing the blocks sub-model design and the D-efficiency factor of each successively fitted blocks sub-model.

Value

| treatments | The treatments included in the design and the replication of each individual treatment |
| design | The design layout showing the allocation of treatment and block design factors to individual plots. |
| treatments_model | The fractional size of the treatment design together with the D-efficiency of that fraction. |
| blocks_model | The blocks sub-model design together with the D-efficiency factor of each successively fitted blocks sub-model. |
| seed | Numerical seed for random number generator. |
| searches | Maximum number of searches in each stratum. |
| jumps | Number of random treatment swaps to escape a local maxima. |

References

Examples

## For optimum results, the number of searches may need to be increased in practice.

```r
## 4 replicates of 12 treatments with 16 nested blocks of size 3
# rectangular lattice see Plan 10.10 Cochran and Cox 1957.
treatments = factor(1:12)
blocks = data.frame(Main = gl(4,12), Sub = gl(16,3))
design(treatments,blocks)$blocks_model
```

## 4 x 12 design for 4 replicates of 12 treatments with 3 plots in each intersection block
## The optimal design is Trojan with known A-efficiency = 22/31 for the intersection blocks
```r
treatments = factor(1:12)
blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48))
design(treatments,blocks)$blocks_model
```

## 4 x 12 design for 4 replicates of 12 treatments with 3 sub-column blocks nested
## as above but showing 3 sub-columns nested within each main column
```r
treatments = factor(1:12)
blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48), subCols = gl(12,1,48))
design(treatments,blocks,searches=200)$blocks_model
```

## 4 x 13 Row-and-column design for 4 replicates of 13 treatments
## Youden design Plan 13.5 Cochran and Cox (1957).
```r
treatments = factor(1:13)
blocks = data.frame(Rows = gl(4,13), Cols = gl(13,1,52))
design(treatments,blocks,searches = 700)
```

## Durban - 272 treatments in a 16 x 34 design with nested rows-and-columns
```r
data(durban)
durban=durban[c(3,1,2,4,5)]
durban=durban[ do.call(order, durban), ]
treatments=data.frame(gen=durban$gen)
Reps = factor(rep(1:2,each=272))
Rows = factor(rep(1:16,each=34))
Col1 = factor(rep(1:4,c(9,8,8,9)),16))
Col2 = factor(rep(1:4,c(5,4,4,4,4,4,4,4,5)),16))
Col3 = factor(rep(1:34,16))
blocks = data.frame(Reps,Rows,Col1,Col2,Col3)
design(treatments,blocks,searches=1)$blocks_model
```

## Finds post-blocked efficiency factors of original design; Durban et al (2003)
```r
blockEfficiencies(treatments,blocks)
```

## differential replication
```r
treatments=factor(c(rep(1:12,2),rep(13,12)))
blocks = data.frame(Main = gl(2,18), Sub = gl(12,3,36))
design(treatments,blocks,searches = 5)
```

## 48 treatments in 2 replicate blocks of size 48 for a 24 x 4 array
## with 2 main rows and 3 main columns the cols factor must precede
## the rows factor otherwise the design will confound one treatment contrast
## in the replicates.rows x columns interactions due to inherent aliasing
treatments=factor(1:48)
blocks = data.frame(Reps = gl(2,48),Cols = gl(3,8,96),Rows = gl(2,24,96))
design(treatments,blocks,searches=5)

### Factorial treatment designs defined by a factorial model equation.

### Main effects of five 2-level factors in a half-fraction in 2/2/2 nested blocks design
### (may require many repeats to find a fully orthogonal solution)
treatments = expand.grid(F1 = factor(1:2), F2 = factor(1:2),
F3 = factor(1:2), F4 = factor(1:2), F5 = factor(1:2))
blocks = data.frame(blocks = gl(4,8))
model = "(F1 + F2 + F3 + F4 + F5)^2"
repeat (z = design(treatments,blocks,treatments_model=model,searches=50)
if ( isTRUE(all.equal(z$blocks_model[3,1]) ) ) break )
print(z)

# Second-order model for five qualitative 2-level factors in 4 randomized blocks
treatments = expand.grid(F1 = factor(1:2), F2 = factor(1:2), F3 = factor(1:2),
F4 = factor(1:2), F5 = factor(1:2))
blocks = data.frame(rows = gl(4,4), cols = gl(4,1,16))
model = "~ F1 + F2 + F3 + F4 + F5"
repeat (z = design(treatments,blocks,treatments_model=model,searches=50)
if ( isTRUE(all.equal(z$blocks_model[2,3,1]) ) ) break )
print(z)

# Quadratic regression for three 3-level numeric factor assuming a 10/27 fraction
treatments = expand.grid(A = 1:3, B = 1:3, C = 1:3)
blocks=data.frame(main=gl(1,10))
model = "~ (A + B + C)^2 + I(A^2) + I(B^2) + I(C^2)"
design(treatments,blocks,treatments_model=model,searches=5)

# First-order model for 1/3rd fraction of four qualitative 3-level factors in 3 blocks
treatments = expand.grid(F1 = factor(1:3), F2 = factor(1:3), F3 = factor(1:3),
F4 = factor(1:3))
blocks = data.frame(main = gl(3,9))
model = "~ F1 + F2 + F3 + F4"
repeat (z = design(treatments,blocks,treatments_model=model,searches=25)

# Second-order model for a 1/3rd fraction of five qualitative 3-level factors in 3 blocks
# (may require many repeats to find a fully orthogonal solution)
treatments = expand.grid(F1 = factor(1:3), F2 = factor(1:3), F3 = factor(1:3),
F4 = factor(1:3), F5 = factor(1:3))
blocks=data.frame(main=gl(3,27))
model = "~ (F1 + F2 + F3 + F4 + F5)^2"
repeat (z = design(treatments,blocks,treatments_model=model,searches=50)
EstEffics

if (isTRUE(all.equal(z$blocks_model[1,3], 1))) break
print(z)

# Second-order model for two qualitative and two quantitative level factors in 4 blocks
# treatments = expand.grid(F1 = factor(1:2), F2 = factor(1:3), V1 = 1:3, V2 = 1:4)
blocks = data.frame(main = gl(4,18))
model = " ~ F1 + F2 + poly(V1,2) + poly(V2,2) + (poly(V1,1) + F1 + F2):poly(V2,1) + F1 + F2"
model = bspline.poly(model, degree = 2)
# expand.grid(F1 = factor(1:2, labels=c("a","b")),
# F3 = factor(1:2, labels=c("a","b")), F4 = factor(1:2, labels=c("a","b")),
# F5 = factor(1:2, labels=c("a","b")), F6 = factor(1:2, labels=c("a","b")),
# F7 = factor(1:2, labels=c("a","b")), F8 = factor(1:2, labels=c("a","b")),
# F9 = factor(1:2, labels=c("a","b")), F10= factor(1:2, labels=c("a","b")),
# F11= factor(1:2, labels=c("a","b")))
blocks = data.frame(main = gl(1,12))
model = " ~ F1 + F2 + F3 + F4 + F5 + F6 + F7 + F8 + F9 + F10 + F11"
model = bspline.poly(model, degree = 2)
design(treatments, blocks, treatments_model=model, searches=5)

---

durban

**Durban example data design**

**Description**


**Usage**

data(durban)

**Format**

An object of class `data.frame` with 544 rows and 5 columns.

---

EstEffics

**D-Efficiency factors**

**Description**

Finds D and A-efficiency for an unstructured treatment set TF with blocks factor BF.
Usage

EstEffics(TF, BF)

Arguments

TF a treatments factor data frame
BF a block factors data frame

Details

efficiency factors of regular block designs

Description

Finds a full column rank model for a treatment formula and a treatment data frame

Usage

fullModel(TF, model_formula)

Arguments

TF is the treatments factor data frame
model_formula is the matrix.model formula

Details

The treatment formula fits a model.matrix for a set of treatments which may be factorial or numeric or a mixture of both. The function will fit the specified model and if there are column dependencies the function will then reduce the column space to full rank using the QR decomposition

Value

Maximal full rank model matrix for fitted model

Examples

# Treatments are two 2-level factors A and B and one 3-level factor V
# Required model is A + B + A:B + A:linear(V) + B:linear(V) + quadratic(V)
# The example shows model formula which appear 'correct' but which over-parameterize
# the model. It is 'reasonable' to expect model.matrix to give a full rank model and
# the QR method will ensure that the fitted model is indeed a full rank model.
# NB The user MUST THEN CHECK to ensure that the fitted model is the required model.
treatments = expand.grid(A = factor(1:2), B = factor(1:2), V = 1:3)

model = " ~ A * B + poly(V,2) + A:poly(V,1) + B:poly(V,1)"
model.matrix(as.formula(model), treatments)
fullModel(treatments, model)

model = " ~ (A + B) * poly(V,1) + poly(V,2) + A:B"
model.matrix(as.formula(model), treatments)
fullModel(treatments, model)

---

**HCF**

*Find hcf*

**Description**

Finds the highest common factor (hcf) of a set of integer numbers greater than zero (Euclidean algorithm)

**Usage**

HCF(v)

**Arguments**

v is the vector of integers for which the hcf is required (must be integers)

**Details**

Finds the hcf of a vector of positive integers which can be in any order

**Value**

hcf

**Examples**

# hcf of vectors of integers
HCF(c(56,77,616))
HCF(c(3,56,77,616))
isPrime

Prime number test

Description
Tests if a given number v is prime and returns TRUE or FALSE

Usage
isPrime(v)

Arguments
v is the number to be tested for primality

Details
Primality of any positive integer based on the fact that all primes except 2 and 3 can be expressed as 6k-1 or 6k+1

Value
logical TRUE or FALSE

Examples
isPrime(731563)
isPrime(7315631)

orthogls
Mutually orthogonal latin square arrays of size v

Description
Constructs a list of orthogonal Latin square arrays of size v.

Usage
orthogls(v)

Arguments
v the dimension of the required MOLS
Details

Returns at least 3 orthogonal latin square arrays of dimension v. The first and second arrays are the rows and columns of a \( v \times v \) square and the third is a Latin square of size \( v \times v \). If \( v \) is prime or prime power in the set 4, 8, 16, 32, 64, 128, 9, 27, 81, 25, 49 there are \( v-1 \) MOLS and \texttt{orthogLS(v)} will return a total of \( v+1 \) arrays. If \( v = 10 \) there are two MOLS and \texttt{orthogLS(10)} will give a total of 4 arrays of size 10 \( \times \) 10.

NB If \( S_1 \) and \( T_1 \) are mutually orthogonal Latin squares of order \( n_1 \) and \( S_2 \) and \( T_2 \) are mutually orthogonal Latin squares of order \( n_2 \) then the product squares \( S_1 \times S_2 \) and \( T_1 \times T_2 \) are orthogonal to each other and have order \( n_1 n_2 \) (not yet implemented)

Value

Treatments A table showing the replication number of each treatment in the design.

References


Examples

```r
## Set of 5 \times 5\ MOLS
orthogLS(5)
```
Index

*Topic data
  durban, 12

A_bound, 3

blockEfficiencies, 4
blocks, 3, 5
blocksdesign (blocksdesign-package), 2
blocksdesign-package, 2

design, 3, 7
durban, 12

EstEffics, 12

fullModel, 13

HCF, 14

isPrime, 15

MOLS, 6

orthogLS, 15