Package 'blocksdesign'

May 28, 2019

Type Package

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

Version 3.5

Date 2019-05-28"

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 model matrix design formula and can be added sequentially, if required. 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 2019-05-28 11:40:03 UTC

R topics documented:

`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 treatments. 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, usually 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 preceding 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 blocks 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 bock 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.

For more details see the 'blocksdesign" vignette: `vignette(package = "blocksdesign")`

References

BATES D., MAECHLER M., BOLKER B., WALKER S. (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1), 1-48. doi:10.18637/jss.v067.i01.

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

John, J. A. and Williams, E. R. (1995). Cyclic and Computer Generated Designs. Chapman and Hall, London.

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 = NULL)
```

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

```
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*p 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)*p 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 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

`Treatments`	A table showing the replication number of each treatment in the design.
`Design`	Data frame giving the optimized block and treatment design in plot order.
`Plan`	Data frame showing a plan view of the treatment design in the bottom level of the design.
`blocks_model`	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.
`seed`	Numerical seed used for random number generator.
`searches`	Maximum number of searches used for each level.
`jumps`	Number of random treatment swaps used to escape a local maxima.

References

Sailer, M. O. (2013). crossdes: Construction of Crossover Designs. R package version 1.1-1. https://CRAN.R-project.org/package=crossdes

Cochran, W.G., and G.M. Cox. 1957. Experimental Designs, 2nd ed., Wiley, New York.

Examples

```
## 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 any feasible combinations of nested or crossed block factors and any feasible linear treatment model.

Usage

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

Arguments

`treatments`	a single treatment factor or data frame containing one or more qualitative or quantitative (numeric) level treatment factors.
`blocks`	a single block factor or 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.

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.

Details

The treatments object is a factor or a data frame containing one or more qualitative or quantitative level treatment factors defining a set of candidate treatments from which the optimized treatment design is selected.

The blocks object is a factor or a data frame containing one or more qualitative level block factors. The default blocks object is a single level factor equal in length to the treatments object. The size of the blocks object defines the required design size.

The treatments_model can be either a single formula for a single design matrix or a compound formula for two or more design matrices. A compound formula contains one or more occurrences of a splitting operator | which splits-off a partial design formula on the left hand side of each |. Assuming the left hand part of each split is a well defined model design formula and replacing all remaining | by + in each partial design formula gives a hierarchical set of design matrices for sequential model fitting. The advantage of sequential model fitting is that it provides improved flexibility for fitting factors or variables of different status or importance and allows a wider range of choices of optimized design for different situations (see examples below).

Treatments are selected from the design candidate set with replacement unless the size of the candidate set exactly equals the size of the required design and the treatments_model is null, in which case the full candidate set is used for the treatment design. This option allows any arbitrary treatment set with any arbitrary treatment replication to be input as the required treatment design. If the size of the candidate set is different from the size of the required design, or if the treatments_model is non-null, the treatment design is optimized by selection with replacement.

The treatment design criterion is the ratio of the generalized variance of the treatment design based on the full treatment candidate set relative to the generalized variance of the treatment design based on the optimized treatment set (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 quantitative level treatments, however, a full factorial design may not provide an optimal design and then the relative efficiency of the optimized design may exceed 1. The efficiency factor is used to compare different optimizations of the same design with the best design having the largest efficiency.

The design algorithm fits the blocks design by sequentially adding the block 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 block main effects versus the block 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, 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 gives a compromise design where all factorial block effects are included in the blocks design but where the main block effects are given greater importance than the block interaction effects.See `vignette(package = "blocksdesign")` for more details.

The blocks design criterion is the D-efficiency of the treatment design without block effects versus the efficiency of the treatment design with block effects. The D-efficiency factor is necessarily less than or equal to one with the most efficient design giving the largest D-efficiency factor. For unstructured treatment designs, the A-efficiency factor is also shown together with an estimated A-efficiency upper-bound, where available.

For more details see `vignette(package = "blocksdesign")`

Value

`treatments`	The treatments included in the design and the replication of each individual treatment taken in de-randomized standard order.
`design`	The design layout showing the randomized allocation of treatments to blocks and plots.
`treatments_model`	The fitted treatment model, the number of model parameters (DF) and the D-efficiency of each sequentially fitted treatment model
`blocks_model`	The blocks sub-model design and the D- and A-efficiency factors of each successively fitted sub-blocks 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

Cochran W. G. & Cox G. M. (1957) Experimental Designs 2nd Edition John Wiley & Sons.

Examples

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

## 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)

## 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
treatments = factor(1:12)
blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48))
```

```
design(treatments,blocks)

## 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
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)

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

## differential replication
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 single factorial treatment model

## Main effects of five 2-level factors in a half-fraction in 2/2/2 nested blocks design
## (may require many 100's of 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(b1 = gl(2,8),b2 = gl(4,4),b3 = gl(8,2))
model=" ~ F1 + F2 + F3 + F4 + F5"
repeat {z = design(treatments,blocks,treatments_model=model,searches=50)
if ( isTRUE(all.equal(z$blocks_model[3,3],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(blocks = gl(4,8))
model = " ~ (F1 + F2 + F3 + F4 + F5)^2"
design(treatments,blocks,treatments_model=model,searches = 10)

# Main effects of five 2-level factors in a half-fraction of
# a 4 x 4 row-and column design.
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=10)

# Quadratic regression for three 3-level numeric factor crossed with a qualitative 2-level factor
treatments = expand.grid(F = factor(1:2), A = 1:3, B = 1:3, C = 1:3)
blocks=data.frame(main=gl(1,18))
model = " ~ F + A + B + C + F:A + F:B + F:C + A:B + A:C + B:C + I(A^2) + I(B^2) + I(C^2)"
design(treatments,blocks,treatments_model=model,searches=5)

# 1st-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"
design(treatments,blocks,treatments_model=model,searches=25)

# 2nd-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)
if ( isTRUE(all.equal(z$blocks_model[1,3],1) ) ) break}
 print(z)

# 2nd-order model for two qualitative and two quantitative level factors in 2 blocks of size 18
treatments = expand.grid(F1 = factor(1:2), F2 = factor(1:3), V1 = 1:3, V2 = 1:4)
blocks = data.frame(main = gl(2,18))
model = " ~ (F1 + F2 + V1 + V2)^2 +  I(V1^2) +  I(V2^2)"
design(treatments,blocks,treatments_model=model,searches=5)

# Plackett and Burman design for eleven 2-level factors in 12 runs
GF = expand.grid(F1 = factor(1:2,labels=c("a","b")), F2 = 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"
design(GF,blocks,treatments_model=model,searches=25)

## Factorial treatment designs defined by sequentially fitted factorial treatment models

## 2 varieties x 3 levels of N x 3 levels of K assuming 1st-order interactions and 12 plots
```

```
## the single stage model gives an unequal 7 + 5 split for the two varieties
## whereas the two stage model forces an equal 6 + 6 split
## NB the two stage model is slightly less efficient than the single stage model
treatments = expand.grid(Variety = factor(rep(1:2)), N = 1:3, K = 1:3)
blocks=data.frame(main=gl(1,12))
treatments_model = " ~  (Variety + N + K)^2  + I(N^2) + I(K^2)"
design(treatments,blocks,treatments_model=treatments_model,searches=10)
treatments_model = " ~ Variety | (Variety + N + K)^2 + I(N^2) + I(K^2)"
design(treatments,blocks,treatments_model=treatments_model,searches=10)

## 8 varieties x 4 levels of N x 4 levels of K assuming 1st-order interactions and 2 blocks
## of size 16, A two stage treatment model gives fully orthogonal blocks with
## no loss of treatment efficiency comapred with a single stage treatment design
treatments = expand.grid(variety = factor(1:8), N = 1:4, K = 1:4)
blocks=data.frame(blocks=gl(2,16,32))
treatments_model = "~ (variety + N + K)^2  + I(N^2) + I(K^2) + I(N^3) + I(K^3)"
design(treatments,blocks,treatments_model=treatments_model,searches=50)
treatments_model = "~variety | (variety + N + K)^2   + I(N^2) + I(N^3) + I(K^2)+ I(K^3)"
design(treatments,blocks,treatments_model=treatments_model,searches=50)

## A 6 x 6 row-and-column design with linear row by linear column interaction.
## Crossed blocks with interactions fitted in the treatments model and additive
## treatments fitted inthe blocks model as a dual design
##  see vignette(package = "blocksdesign") for further discussion
## may require many separate attempts to get the best overall design efficiency
LS_grid   = expand.grid(rows=factor(1:6), cols=factor(1:6))
blocks = data.frame(varieties=factor(rep(1:6,6)))
lin_rows = as.numeric(levels(LS_grid$rows))[LS_grid$rows]
lin_cols = as.numeric(levels(LS_grid$cols))[LS_grid$cols]
latin_sq = "~  rows | cols + lin_rows:lin_cols "
design(LS_grid,blocks,latin_sq,searches=2000)
```

---

durban *Durban example data design*

Description

Actual layout used by DURBAN, M., HACKETT, C., MCNICOL, J., NEWTON, A., THOMAS, W., & CURRIE, I. (2003). The practical use of semi-parametric models in field trials, Journal of Agric Biological and Envir Stats, 8, 48-66.

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 treatmnent 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

fullrankModel *Full rank model*

Description

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

Usage

```
fullrankModel(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)
fullrankModel(treatments,model)

model = " ~ (A + B) * poly(V,1) + poly(V,2)  + A:B"
model.matrix(as.formula(model),treatments)
fullrankModel(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 zize 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 x v square and the third is a Latin square of size v x 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 `orthogLS(v)` will return a total of v+1 arrays. If v = 10 there are two MOLS and `orthogLS(10)` will give a total of 4 arrays ofsize 10 x 10.

NB If S1 and T1 are mutually orthogonal Latin squares of order n1 and S2 and T2 are mutually orthogonal Latin squares of order n2 then the product squares S1xS2 and T1xT2 are orthogonal to each other and have order n1n2 (not yet implemented)

Value

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

References

Sailer, M. O. (2013). crossdes: Construction of Crossover Designs. R package version 1.1-1. https://CRAN.R-project.org/package=crossdes

Examples

```
## Set of 5 x 5 MOLS
orthogLS(5)
```

Index