Package 'blocksdesign'

April 7, 2021

Type Package

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

Version 4.9

Date 2021-04-07"

Author R. N. Edmondson.

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

Depends R (>= 3.1)

Description Constructs treatment and block designs for linear treatment models with crossed or nested block factors. The treatment design can be any feasible linear model and the block design can be any feasible combination of crossed or nested block factors. The block design is a sum of one or more block factors and the block design is optimized sequentially with the levels of each successive block factor optimized conditional on all previously optimized block factors. D-optimality is used throughout except for square or rectangular lattice block designs which are constructed algebraically using mutually orthogonal Latin squares. Crossed block designs with interaction effects are optimized using a weighting scheme which allows for differential weighting of first and second-order block effects. 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. Edmondson, R.N. Multi-level Block Designs for Comparative Experiments. JABES 25, 500–522 (2020) <doi:10.1007/s13253-020-00416-0>.

License GPL (>= 2)

URL <doi:10.1007/s13253-020-00416-0>

Imports plyr,PolynomF

LazyData true

RoxygenNote 7.1.1

Encoding UTF-8

Suggests R.rsp

VignetteBuilder R.rsp

NeedsCompilation no

Repository CRAN

Date/Publication 2021-04-07 18:20:06 UTC

R topics documented:

blocksdesign-package 2
A_bound 3
blocks 4
design 6
durban 11
fraction 12
GraecoLatin 14
HCF 15
isPrime 16
isPrimePower 17
MOLS 17

Index **20**

`blocksdesign-package` *Blocks design package*

Description

The `blocksdesign` package provides functionality for the construction of block and treatment designs for general linear models.

Details

Randomized complete blocks are the designs of choice for small experiments with few treatments. For large experiments with many treatments, however, a single set of complete blocks may not be adequate and then sub-division into smaller nested blocks may be required. Block designs with a single level of nesting are widely used but a single level of nesting may be inadequate for very large experiments with many treatments. `blocksdesign` provides for the construction of designs with multiple levels of nesting down to any feasible depth of nesting.

Sometimes block designs for the control of variability in two or more dimensions are required and `blocksdesign` can also build crossed block designs allowing for both additive and interactive crossed block effects simultaneously.

The `blocksdesign` package has two main functions:

i) `blocks`: This is a simple recursive function for nested blocks for unstructured treatments. The function generates designs for treatments with arbitrary levels of replication and with arbitrary depth of nesting where blocks sizes are assumed to be as equal as possible for each level of nesting. Special square and rectangular lattice designs (see Cochran and Cox 1957) are constructed algebraically from mutually orthogonal Latin squares (MOLS). The outputs from the `blocks` function include a data frame showing the allocation of treatments to blocks 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 for the bottom level of the design is also included in the output.

ii) design: This is a general purpose function for linear models with qualitative or quantitative level treatment factors and qualitative level block factors. The function finds a D-optimal or near D-optimal design for a specified treatment model and then finds a conditional D-optimal or near D-optimal block design for that choice of treatment design. The design algorithm builds the blocks design by sequentially adding blocks factors where each blocks factor is optimized conditional on all previously added blocks factors. 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.

Other available functions are A_bound, which finds upper A-efficiency bounds for regular block designs, MOLS, which constructs sets of mutually orthogonal prime-power Latin squares (MOLS), GraecoLatin, which constructs mutually orthogonal Graeco-Latin squares not necessarily prime-power, isPrime, which tests an integer for primality, isPrimePower, which factorizes prime powers and HCF, which finds the highest common factor (hcf) for a set of positive integer numbers.

For further explanation see Edmondson (2020) and vignette(package = "blocksdesign").

References

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

Edmondson, R.N. Multi-level Block Designs for Comparative Experiments. JABES 25, 500–522 (2020).

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 with equi-replicate treatments and equal block sizes (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 only. 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)
```

`blocks` *Block designs for unstructured treatment sets*

Description

Constructs randomized multi-level nested block designs for unstructured treatment sets.

Usage

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

Arguments

`treatments`	the total required number of treatments partitioned into equally replicated treatment sets.
`replicates`	the replication numbers of the equally replicated treatment sets.
`blocks`	the number of nested blocks in each level of nesting from the top level down.
`searches`	the maximum number of local optima searched for a design optimization.
`seed`	an integer initializing the random number generator.
`jumps`	the number of pairwise random treatment swaps used to escape a local maxima.

Details

Constructs randomized multi-level nested block designs for any arbitrary number of unstructured treatments and any arbitrary feasible depth of nesting.

`treatments` is a partition of the number of treatments into equi-replicate treatment sets.

`replicates` is a set of replication numbers for the equi-replicate treatment sets.

blocks are the nested blocks levels in decreasing order of block size where each level defines the number of blocks nested within the blocks of the preceding level. The top-level block is assumed to be single super-block containing a full set of plots. The algorithm finds block sizes automatically for each level of nesting and the block sizes within each level of nesting will never differ by more than a single plot.

Unreplicated treatments are allowed and any simple nested block design can be augmented by any number of unreplicated treatments using the treatments and replicates formula. However, it may be preferable to find an efficient blocked design for the replicated treatment sets and then to add the unreplicated treatments heuristically.

The blocks function constructs all block designs algorithmically except for certain special block designs for r replicates of v x v treatments or r-1 replicates of v x (v-1) treatments in blocks of size v. Provided that a set of r-1 mutually orthogonal Latin squares of size v x v exists, these designs are constructed algebraically and are guaranteed to achieve optimality. See squarelattice and rectlattice for information about which design sizes are constructed algebraically.

Value

Replication	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

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.

# Completely randomized design for 6 treatments with 2 replicates and 1 control with 4 replicates
blocks(treatments=list(6,1),replicates=list(2,4))

# 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=list(4,4))

# 3 treatments x 2 replicates + 2 treatments x 4 replicates in two complete randomized blocks
blocks(treatments=list(3,2),replicates=list(2,4),blocks=2)
```

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

# as above but with 20 additional single replicate treatments, one single treatment per sub-block
blocks(treatments=list(50,20),replicates=list(4,1),blocks=list(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,list(2,2,2,2))

# 64 treatments x 4 replicates with 4 main blocks, 8 nested sub-blocks of size 8
# (lattice), 16 nested sub-sub blocks of size 4 and 32 nested sub-sub-sub blocks of size 2
 blocks(64,4,list(4,8,2,2))

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

`design` *General block and treatment designs.*

Description

Constructs D-optimal block and treatment designs for any feasible linear treatment model and any feasible combination of block factors.

Usage

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

Arguments

`treatments`	a candidate set of treatments for any combination of qualitative or quantitative level factors.
`blocks`	a data frame of block factors.
`treatments_model`	a list containing one or more nested treatment model formula.

weighting	a weighting factor between 0 and 1 for weighting the 2-factor interaction effects of factorial blocks.
searches	the maximum number of local optima searched at each stage of an optimization.
seed	an integer initializing the random number generator.
jumps	the number of pairwise random treatment swaps used to escape a local maxima.

Details

treatments a factor or data frame or list of generators for the candidate set of treatments. The candidate treatment set can include both quantitative and qualitative level factors and the required design is selected from the candidate treatment set without replacement. The number of times a treatment can be included in a design depends on the replication of that treatment in the candidate set therefore increasing the treatment replication in the candidate set allows increased replication for individual treatments.

blocks a data frame of nested or crossed block factors. The length of the block factors defines the required size of the treatment design. the treatment design is optimized for the required blocks design by maximising the determinant of the block-adjusted treatment information matrix (D-optimality. For crossed block designs, the information matrix is based on a weighted combination of block main effects and two-factor interaction effects where the information on the interaction effects is down-weighted by a coefficient $0 < w < 1$ where $w = 0$ gives blocks main effects only whereas $w = 1$ gives the full two-factor blocks interaction model. This process ensures that factorial main effects can be given more importance than the interaction effects in the optimization of a crossed blocks design.

treatments_model a character vector containing one or more treatments model formula. The treatment models are optimized sequentially for each model formula in turn assuming the treatments of any previously fitted models are held constant. Sequential fitting provides improved flexibility for fitting treatment factors or variables of different status or importance (see examples below). The design criterion for each treatment model is maximization of the determinant of the information matrix of that treatment model conditional on the the information matrices of all previously fitted treatment models.

The D-optimal treatment design efficiency is the ratio of the generalized variance of the full candidate treatment model relative to the generalized variance of the optimized design. The efficiency will be less than or equal to 1 for factorial models but may exceed 1 for polynomial models.

The design outputs include a table of efficiency factors for first and second-order factorial block effects and comparison of the efficiency factors for different choices of w can be used to find a good compromise design for fitting both main block and interaction block effects.

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

Value

Replication	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 in standardized block order.
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

Gupta, S. C., and B. Jones. Equireplicate Balanced Block Designs with Unequal Block Sizes. Biometrika, vol. 70, no. 2, 1983, pp. 433–440.

Examples

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

## 4 replicates of 12 treatments with 16 nested blocks of size 3
## giving a rectangular lattice: see Plan 10.10 Cochran and Cox 1957.

blocks = data.frame(Main = gl(4,12), Sub = gl(16,3))
treatments = data.frame(treatments = gl(12,1,48))
Z=design(treatments, blocks)
print(Z)

## 3 replicates of 15 treatments in 3 main blocks with two sets of
## nested blocks and one control treatment

blocks=data.frame( Main = gl(3,18,54),Sub1 = gl(9,6,54),Sub2 = gl(27,2,54))
treatments=factor(rep(c(1:15,rep("control",3)),3))
Z=design(treatments,blocks)
print(Z)
incid1=table(interaction(Z$Design$Main,Z$Design$Sub1,lex.order = TRUE),Z$Design$treatments)
crossprod(incid1) # print pairwise concurrences within Sub1 blocks
incid2=table(interaction(Z$Design$Main,Z$Design$Sub2,lex.order = TRUE),Z$Design$treatments)
crossprod(incid2) # print pairwise concurrences within Sub2 blocks


## 4 x 12 design for 4 replicates of 12 treatments with 3 plots in each intersection block
## and  3 sub-columns nested within each main column. The optimal row-and column design
## is Trojan with A-efficiency = 22/31 for the intersection blocks

blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48), subCols = gl(12,1,48))
treatments = data.frame(treatments =gl(12,1,48))
Z=design(treatments,blocks)
print(Z)
incid=table(interaction(Z$Design$Rows,Z$Design$Cols,lex.order = TRUE),Z$Design$treatments)
crossprod(incid) # print pairwise concurrences within blocks


## 4 x 13 Row-and-column design for 4 replicates of 13 treatments
## Youden design Plan 13.5 Cochran and Cox (1957).
```

```
blocks = data.frame(Rows = gl(4,13), Cols = gl(13,1,52))
treatments = data.frame(treatments= gl(13,1,52))
Z=design(treatments,blocks,searches = 700)
print(Z)
incid=table(Z$Design$Cols,Z$Design$treatments)
crossprod(incid) # print pairwise concurrences of treatments within column blocks (BIB's)


## 48 treatments in 2 replicate blocks with 2 nested rows in each replicate and 3 main columns

blocks = data.frame(Reps = gl(2,48), Rows = gl(4,24,96), Cols = gl(3,8,96))
treatments = data.frame(treatments=gl(48,1,96))
Z=design(treatments,blocks,searches=5)
print(Z)

## 48 treatments in 2 replicate blocks with 2 main columns
## The default weighting gives non-estimable Reps:Cols effects due to inherent aliasing
## Increased weighting gives estimable Reps:Cols effects but non-orthogonal main effects

blocks = data.frame(Rows = gl(2,48), Cols = gl(2,24,96))
Z1=design(treatments=gl(48,1,96),blocks,searches=5)
print(Z1)
Z2=design(treatments=gl(48,1,96),blocks,searches=5,weighting=.9)
print(Z2)


## Equireplicate balanced designs with unequal block sizes. Gupta & Jones (1983).
## Check for equality of D and A-efficiency in optimized design

t=factor(c(rep(1:12,each=7)))
b=factor(c(rep(1:12,each=6),rep(13:18,each=2)))
Z=design(t,b,searches=100)$Blocks_model # max efficiency = 6/7
print(Z)


t=factor(c(rep(1:14,each=8)))
b=factor(c(rep(1:14,each=4),rep(15:21,each=8)))
Z=design(t,b,searches=500)$Blocks_model # max efficiency = 7/8
print(Z)


t=factor(c(rep(1:16,each=7)))
b=factor(c(rep(1:16,each=4),rep(17:22,each=8)))
Z=design(t,b,searches=1000)$Blocks_model # max efficiency = 6/7
print(Z)


t=factor(c(rep(1:18,each=7)))
b=factor(c(rep(1:18,each=6),rep(19:24,each=3)))
Z=design(t,b,searches=500)$Blocks_model # max efficiency = 6/7
print(Z)
```

```
## Factorial treatment designs defined by a factorial treatment model

## Main effects of five 2-level factors in a half-fraction in 2/2/2 nested blocks design
## The algorithmic search method is likely to require a very long search time to reach the
## known orthogonal solution for this regular fractional factorial design

treatments = list(F1 = gl(2,1), F2 = gl(2,1),F3 = gl(2,1), F4 = gl(2,1), F5 = gl(2,1))
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 with two
# nested sub-blocks in each main plot
treatments = list(F1 = gl(2,1), F2 = gl(2,1),F3 = gl(2,1), F4 = gl(2,1), F5 = gl(2,1))
blocks = data.frame(main = gl(4,8),sub=gl(8,4))
model = ~ (F1 + F2 + F3 + F4 + F5)^2
Z=design(treatments,blocks,treatments_model=model,searches = 10)
print(Z)

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

treatments = list(F1 = gl(2,1), F2 = gl(2,1),F3 = gl(2,1), F4 = gl(2,1), F5 = gl(2,1))
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 = list(A = 1:3, B = 1:3, C = 1:3)
blocks=factor(rep(1,10))
model = ~ ( A + B + C)^2 + I(A^2) + I(B^2) + I(C^2)
Z=design(treatments,blocks,treatments_model=model,searches=10)
print(Z)

## response surface designs with doubled treatment candidate sets
## allowing duplicated design points

## two treatment factor design showing double replication on the
## four corner points and double replication on the centre point
treatments = list( V1 = 1:3, V2 = rep(1:3,2))
blocks=factor(rep(1,14))
model =  ~ (V1 + V2)^2 + I(V1^2) + I(V2^2)
design(treatments,blocks,treatments_model=model,searches=10)

## three treatment factor design with eight corner points, 12 edge points and 1 centre
## point. All 21 design points are accounted for leaving nothing for extra replication.
treatments = list( V1 = 1:3, V2 = 1:3, V3 = rep(1:3,2))
```

```
blocks=factor(rep(1,21))
model =  ~ (V1 + V2 + V3)^2 + I(V1^2) + I(V2^2) + I(V3^2)
design(treatments,blocks,treatments_model=model,searches=20)

## as above but with extra replication on each corner point and on the centre point
treatments = list( V1 = 1:3, V2 = 1:3, V3 = rep(1:3,2))
blocks=factor(rep(1,30))
model =  ~ (V1 + V2 + V3)^2 + I(V1^2) + I(V2^2) + I(V3^2)
design(treatments,blocks,treatments_model=model,searches=20)

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

## 4 varieties by 3 levels of N by 3 levels of K assuming degree-2 treatment
## interaction effects and two blocks of 12 plots
## the single stage model gives an unequal split for the replication of the four varieties
## which may be undesirable whereas the two stage model forces an equal split of 6 plots
## per variety. The single stage model is slightly more efficient than the two stage model
## (1.052045 versus 0.9761135 standardized relative to the full factorial design)  but
## in this example global D-optimality does not give the most suitable design structure.

treatments = list(Variety = factor(1:4), N = 1:3, K = 1:3)
blocks = data.frame(main=gl(2,12))
treatments_model = ~ (Variety + N + K)^2  + I(N^2) + I(K^2)
Z1 = design(treatments,blocks,treatments_model=treatments_model,searches=10)
print(Z1)
treatments_model = list( ~ Variety , ~ Variety + (Variety + N + K)^2 + I(N^2) + I(K^2))
Z2 = design(treatments,blocks,treatments_model=treatments_model,searches=10)
print(Z2)
```

---

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. This data set was obtained from the durban.rowcol agridat example in package 'agridat' Kevin Wright (2020). agridat: Agricultural Datasets. R package version 1.17.

Usage

```
data(durban)
```

Format

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

`fraction` *Optimum treatment set from a candidate set of treatments*

Description

Finds an optimum set of treatments from a candidate set of treatments for any arbitrary treatments design formula.

Usage

```
fraction(
  treatments,
  size,
  treatments_model = NULL,
  restriction_model = NULL,
  searches = 50
)
```

Arguments

`treatments`	is a data frame or a list containing a candidate set of factorial treatments.
`size`	is the required number of treatments in the fractional set of treatments.
`treatments_model`	is a model formula for the required treatments design.
`restriction_model`	is a model formula which is a subset of the `treatments_model` formula and which fixes those treatment factors contained in the restriction model formula.
`searches`	are the maximum number of searches for selecting the best optimization.

Details

The candidate set `treatments` will normally contain one or more complete sets of treatment replicates. The algorithm re-arranges the rows of the `treatments` set to ensure that the first `size` rows of the optimized `treatments` set contains the optimized treatment fraction. The maximum replication of any treatment is the number of times that treatment occurs in the candidate treatment set and for a polynomial response surface design extra replication of the candidate set may be necessary to allow for differential replication of the design points. The design is optimized with respect to the `treatments_model` conditional on the treatment factors in the `restriction_model` being held constant. The `restriction_model` must be a subset of the full `treatments_model` otherwise the design will be fully fixed and no further optimization will be possible. Fitting a non-null `restriction_model` allows sequential optimization with each successively `treatments_model` optimized conditional on all previously optimized models. The D-optimal efficiency of the design for the optimized treatment set is calculated relative to the D-optimal efficiency of the design for the candidate treatment set.

The default `treatments_model` parameter is an additive model for all treatment factors.

Value

A list containing:

- "TF" An optimized treatment fraction of the required `size`
- "fullTF" The full candidate set of treatments but with the first `size` rows containing the optimized treatment fraction
- "Efficiency" The D-efficiency of the optimized treatment fraction relative to the full candidate set of treatments

See Also

`design`

Examples

```
#' ## Plackett and Burman (P&B) type design for eleven 2-level factors in 12 runs
## NB. The algorithmic method is unlikely to succeed for larger P&B type designs.

GF = list(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")) )
model = ~ F1 + F2 + F3 + F4 + F5 + F6 + F7 + F8 + F9 + F10 + F11
Z=fraction(GF,size=12,treatments_model=model,searches=100)
print(Z$TF)
print(Z$Efficiency)
round(crossprod(scale(data.matrix(Z$TF))),6)

## Factorial treatment designs defined by sequentially fitted factorial treatment models
## 4 varieties by 3 levels of N by 3 levels of K assuming degree-2 treatment model in 24 plots.
## The single stage model gives an unequal split for the replication of the four varieties
## whereas the two stage model forces an equal split of 6 plots per variety.
## The single stage model is slightly more efficient overall (about 1.052045 versus 1.043662)
## but unequal variety replication is undesirable if all varieties are equally important.

## model terms
treatments = list(Variety = factor(1:4), N = 1:3, K = 1:3)
variety_model = ~ Variety
full_model = ~ (Variety + N + K)^2  + I(N^2) + I(K^2)

## single stage model
opt_full_treatments = fraction(treatments,24,full_model,searches=10)
opt_full_treatments$Efficiency
table(opt_full_treatments$TF[,1]) # variety replication

## two stage model
opt_var_treatments  = fraction(treatments,24,variety_model,searches=10)
opt_full_treatments = fraction(opt_var_treatments$fullTF,24,full_model,variety_model,searches=10)
opt_full_treatments$Efficiency
```

```
table(opt_full_treatments$TF[,1]) # variety replication
```

GraecoLatin *Graeco-Latin squares*

Description

Constructs mutually orthogonal Graeco-Latin squares for the following N:

i) any odd valued N

ii) any prime-power N = p**q where p and q can be chosen from

prime p	maximum q
2	13
3	8
5	6
7	5
11	4
13 17 19 23	3
29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97	2
Any prime >97	1

iii) any even valued N <= 30 except for 6 or 2

Usage

```
GraecoLatin(N)
```

Arguments

N	any suitable integer N

Details

Plans are given for pairs of MOLS classified by rows and columns. The output is a single data frame of size $p * *qx(r+2)$ for the required set of MOLS with a column for the rows classification, a column for the columns classification and a column for each treatment set from the required set of MOLS.

Also see the function MOLS which will generate complete sets of MOLS for prime-power design sizes.

Value

Data frame of factor levels for rows, columns and treatment sets

References

Street, A. P. & Street, D. J. (1987). Combinatorics of Experimental Design, Chapters 6 and 7. Clarendon Press, Oxford.

See Also

MOLS

Examples

```
X=GraecoLatin(8)
table(X[,3],X[,4])
X=GraecoLatin(9)
table(X[,3],X[,4])
X=GraecoLatin(32)
table(X[,3],X[,4])
```

HCF *Finds hcf of any set of positive integers*

Description

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

Usage

```
HCF(...)
```

Arguments

...	any set of positive integers, in any order, for which the hcf is required.

Details

Finds the hcf of any set of positive integers which can be in any order.

Value

hcf

References

Euclidean algorithm, see:https://mathworld.wolfram.com/EuclideanAlgorithm.html

Examples

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

`isPrime` *Prime number test*

Description

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

Usage

```
isPrime(v)
```

Arguments

v	the number to be tested for primality

Details

Tests for the primality of any positive integer using the fact that all primes except 2 and 3 can be expressed as 6k-1 or 6k+1 for integer k.

Value

logical TRUE or FALSE

Examples

```
isPrime(731563)
isPrime(7315631)
isPrime(31**2)
```

isPrimePower — *Finds a prime power solution for N, if available.*

Description

Tests if a given number N is a prime power and returns either the base prime p and power q or p = 0 and q = 0.

Usage

```
isPrimePower(N)
```

Arguments

N	the number to be tested for primality

Details

Finds the smallest integral solution for s = N**(1/i), which gives the smallest s such that s**i = N. Then, if s is a prime, the number N is a prime power with p = s and q = i.

Value

Returns the base prime p and the power q if N is a prime power; otherwise returns p = 0 and q = 0.

Examples

```
isPrimePower(N=13**5)
isPrimePower(N=25**5) ## 25 is prime-power
isPrimePower(N=20**5) ## 20 is not prime-power
```

MOLS — *Prime power MOLS from finite fields*

Description

Constructs r sets of mutually orthogonal Latin squares (MOLS) of dimension p**q for prime p and integer power q where r < p**q. Memory issues mean that the maximum size of the exponent q for specific p is restricted to the values shown in the table below:

prime p	maximum q
2	13
3	8
5	6

7	5
11	4
13 17 19 23	3
29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97	2
Any prime >97	1

Usage

```
MOLS(p, q, r = 1)
```

Arguments

p	is any suitable integer base
q	is any suitable integer exponent
r	is any suitable number of squares

Details

Generates MOLS by cyclic permutation of a basic Latin square constructed from a vector of ordered elements of a prime-power finite field of size p**q (see Chapter 1 of Raghavarao 1971).

The primitive polynomials for the MOLS generated by this package were extracted from the Table of Primitive Polynomials given in the Supplement to Hansen and Mullen (1992).

The output is a single data frame of size $p**qx(r+2)$ for the required set of MOLS with a column for the rows classification, a column for the columns classification and separate columns for each treatment set of the required set of squares.

Also see the function `GraecoLatin` which will generate pairs of MOLS for a range of non-prime power design sizes v**2 including all odd values of v and any even valued v <= 30 except for 6 or 2.

Value

Data frame of factor levels for rows, columns and treatment sets

References

Hansen, T. & Mullen, G. L. (1992) Primitive polynomials over finite fields, Mathematics of Computation, 59, 639-643 and Supplement.

Raghavarao D. (1971) Constructions and Combinatorial Problems in Design of Experiments, Dover Publications, Inc. Section 1.3

See Also

`GraecoLatin`

Examples

```
MOLS(2,3,7) # Seven MOLS of size 8 x 8
MOLS(3,2,4) # Four MOLS of size 9 x 9
MOLS(3,3,4) # Four MOLS of size 27 x 27
MOLS(23,2,2) # Two MOLS of size 529 x 529
```

Index

∗ **data**
durban, 11

A_bound, *3*, 3

blocks, *2*, 4
blocksdesign *(*blocksdesign-package*)*, 2
blocksdesign-package, 2

design, *3*, 6, *13*
durban, 11

fraction, 12

GraecoLatin, *3*, 14, *18*

HCF, *3*, 15

isPrime, *3*, 16
isPrimePower, *3*, 17

MOLS, *3*, *15*, 17

rectlattice, *5*

squarelattice, *5*