Package 'blocksdesign'

October 12, 2020

Type Package

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

Version 4.5

Date 2020-10-09"

Author R. N. Edmondson.

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

Depends R (>= 3.1.0)

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 (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 2020-10-12 21:30:03 UTC

R topics documented:

`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 block designs are usually the designs of choice for small experiments but for large experiments, sub-division of complete blocks into smaller incomplete blocks can provide improved precision on within-block treatment comparisons. Incomplete block designs with a single level of nesting are widely used but for large experiments with many plots, a single level of nesting may not be adequate for the control of variability over a range of scales of measurement. Multi-level block designs with a hierarchy of nested block sizes can be valuable for accommodating block variability over the full range of scales of measurement and the `blocksdesign` algorithm provides for the recursive nesting of block factors down to any feasible depth of nesting.

Sometimes a factorial blocking system with two or more sets of crossed factorial block factors is needed for the control of variability in two or more dimensions simultaneously and the `blocksdesign` algorithm can also be used to build crossed block designs by sequential addition of block factors in a defined order of fitting.

Functionality

`blocksdesign` has two main 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. The input requires the number of blocks for each level of nesting and the algorithm automatically finds block sizes that are as equal as possible for each level of nesting. Special block designs including square and rectangular lattice designs (see Cochran and Cox 1957) are constructed algebraically from MOLS or Graeco-Latin squares. 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 linear models for qualitative and quantitative level treatment factors and for qualitative level block factors. The function first finds a D-optimal or near D-optimal treatment design and then finds a D-optimal or near D-optimal block design conditional on that choice of treatment design. The blocks `design` algorithm builds the blocks design by sequentially adding `blocks` factors where each block factor is optimized conditional on all previously added block factors remaining constant. 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 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 (2020). https://link.springer.com/article/1
020-00416-0

`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)
```

---

blocks *Block designs for unstructured treatment sets*

Description

Constructs randomized nested block designs for unstructured treatment sets.

Usage

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

Arguments

treatments	the required number of treatments partitioned into equally replicated treatment sets.
replicates	the replication number for each partitioned treatment set.
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 nested block designs for any arbitrary number of unstructured treatments and any arbitrary feasible depth of nesting.

`treatments` is a set of numbers that partitions the total number of treatments into equi-replicate treatment sets.

`replicates` is a set of treatment replication numbers that defines the replication number of each equi-replicate `treatments` set.

`blocks` is a sequence of nested block levels in descending order of block sizes from largest to smallest where each block level is the number of blocks nested within each preceding block (assuming a single top-level super-block). The block sizes will automatically be as nearly equal as possible and will never differ in size by more than a single plot for any particular level of nesting.

Unreplicated treatments are allowed and any simple nested block design can be augmented by any number of unreplicated treatments using the `treatments` and `replicates` formulae. However, it will usually be preferable to find an efficient block design for just the replicated treatment sets and then to add the unreplicated treatments individually by hand.

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 are square lattices and these designs are constructed automatically from Latin squares or MOLS.

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 are rectangular lattices and these designs are constructed automatically by reducing an appropriate algebraic square lattice, see Cochran and Cox, Experimental Designs, 2nd Edition, Page 417 (Shrikhande method).

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

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 with 4 replicates and 1 control treatment with 8 replicates
# the blocks automatically default to 4 complete randomized blocks each of size 14
blocks(treatments=list(12,1),replicates=list(4,8))

# 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,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=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 nested blocks of size 8 (lattice square)
blocks(64,4,list(4,8))

# 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 general D-optimal designs for any feasible linear treatment model with any feasible combinations of block factors.

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 for the candidate set of treatment factor combinations assuming any combination of qualitative or quantitative factor levels.
blocks	a single block factor or data frame for the required combinations of block factors in the required order of fitting assuming quantitative block factor levels only.
treatments_model	a character vector 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 is a factor or data frame containing one or more qualitative or quantitative level treatment factors defining the candidate treatment set. The required treatment design is selected from the candidate treatment set without replacement.

i) If the candidate set is the same size as the required design, the full candidate set will be selected.

ii) If the candidate set is larger than the required design, a subset of the candidate set will be selected by optimizing the treatment D-optimality criterion.

iii) If the candidate set is smaller than the required design size, the candidate set will be replicated until the candidate set equals or exceeds the required design size.

blocks is a factor or data frame containing one or more qualitative level block factors, added sequentially. If the blocks are fully nested or if the design has additive crossed blocks, each added block factor is optimized by swaps made within the levels of all previously added blocks. If, however, a design has crossed blocks with estimable interaction effects, the relative importance of the main factorial block effects versus the 2-factor block interaction effects is weighted by a parameter w in the range 0 to 1.

i) If w = 0, the design is a simple additive block effects.

ii) If w = 1, the design is a simple interaction blocks design for the crossed blocks interaction effects

iii) If 0 < w < 1, the 2-factor block interaction effects are weighted relative to the additive block effects with the importance of the interaction effects assumed to increase as w increases from 0 to 1.

The length of the `blocks` object defines the total number of plots in the design.

`treatments_model` is a character vector containing one or more nested treatments formula where each model formula, taken in order, is optimized with the treatment factors of all previously optimized model formula remaining constant. Sequential model fitting provides improved flexibility for fitting factors or variables of different status or importance (see examples below).

The treatment design criterion for each treatment model is the generalized variance of the information matrix for that design and the 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.

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

## 3 replicates of 15 treatments in 3 main blocks with 3 nested blocks and one control treatment
blocks=data.frame( Main = gl(3,18,54),Sub = gl(9,6,54))
treatments=factor(rep(c(1:15,rep("control",3)),3),levels = c(1:15,"control") )
Z=design(treatments,blocks)
```

```
incid=table(interaction(Z$Design$Main,Z$Design$Sub,lex.order = TRUE),Z$Design$treatments)
Z # print design
incid # print incidences of treatments in blocks
crossprod(incid) # print pairwise concurrences within 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
blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48))
Z=design(treatments =gl(12,1,48),blocks)
incid=table(interaction(Z$Design$Rows,Z$Design$Cols,lex.order = TRUE),Z$Design$treatments)
Z # print design
incid # print incidences of treatments in blocks
crossprod(incid) # print pairwise concurrences within blocks
## as above but showing 3 sub-columns nested within each main column
blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48), subCols = gl(12,1,48))
Z=design(treatments = gl(12,1,48),blocks,searches=200)
Z # print design

## 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))
Z=design(treatments = gl(13,1,52),blocks,searches = 700)
incid=table(Z$Design$Cols,Z$Design$treatments)
Z # print design
crossprod(incid) # print pairwise concurrences of treatments within column blocks (BIB's)
tcrossprod(incid) # print pairwise concurrences of column blocks within treatments (Dual design)

## 48 treatments in 2 replicate blocks with 2 nested rows in each replicate and 3 main columns
##  (Reps/Rows) x Cols
blocks = data.frame(Reps = gl(2,48), Rows = gl(4,24,96), Cols = gl(3,8,96))
design(treatments=gl(48,1,96),blocks,searches=5)

## 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(Reps = gl(2,48), Cols = gl(2,24,96))
design(treatments=gl(48,1,96),blocks,searches=5)
design(treatments=gl(48,1,96),blocks,searches=5,weighting=.9)

## 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 100's of repeats to find a fully orthogonal solution - a VERY long wait!)
treatments = expand.grid(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
treatments = expand.grid(F1 = gl(2,1), F2 = gl(2,1),F3 = gl(2,1), F4 = gl(2,1), F5 = gl(2,1))
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 = 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 = 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 = gl(3,1), F2 = gl(3,1), F3 = gl(3,1), F4 = gl(3,1))
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 = gl(3,1), F2 = gl(3,1),F3 = gl(3,1), F4 = gl(3,1), F5 = gl(3,1))
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"
```

```
D=design(GF,blocks,treatments_model=model,searches=25)
round(crossprod(scale(data.matrix(D$Design)[,-1])),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
## 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 appears slightly more efficient but
## in this example global D-optimality does not give the most suitable design structure.
treatments = expand.grid(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)"
design(treatments,blocks,treatments_model=treatments_model,searches=10)
treatments_model = c(" ~ Variety" ," ~ Variety + (Variety + N + K)^2 + I(N^2) + I(K^2)")
design(treatments,blocks,treatments_model=treatments_model,searches=10)
```

`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.

`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])
X=GraecoLatin(97)
```

```
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(10000)
```

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. https://www.jstor.org/stable/2153093?seq=1

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

See Also

`GraecoLatin`

Examples

```
MOLS(2,3,1) # Single Latin square of size 8 x 8
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