Package ‘blocksdesign’

May 6, 2020

Type Package

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

Version 4.0

Date 2020-05-01

Author R. N. Edmondson.

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

Depends R (>= 3.1.0)

Description Constructs D-optimal or near D-optimal treatment and block designs for linear treatment models with crossed or nested block factors. The treatment design can be any arbitrary linear model defined by a treatment model formula and the block design can be any feasible combination of crossed or nested block factors. The block design factors are optimized sequentially and the levels of each successive block factor are optimized within the levels of each preceding block factor. Crossed block designs with non-singular interaction effects are optimized using a weighting scheme that 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.

License GPL (>= 2)

Imports lme4, plyr, PolynomF

LazyData true

RoxygenNote 7.1.0

Encoding UTF-8

Suggests R.rsp

VignetteBuilder R.rsp

NeedsCompilation no

Repository CRAN

Date/Publication 2020-05-06 16:30:06 UTC
blocksdesign-package

Description

The blocksdesign package provides functionality for the construction of 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 blocks 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 are commonly used for the control of block effects in two dimensions simultaneously.

Recursive nesting builds structure into the block design at the design stage and an appropriate set of block effects for the control of positional effects can be selected at the analysis stage by the use of selection criteria such as the AIC statistic. See Burnham and Anderson (2002).

Functionality

blocksdesign has two main functions:
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. 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.

design: This is a general purpose function for arbitrary linear treatment designs and arbitrary linear block designs with qualitative factor levels. 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 for that 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. 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


---

A_bound

Description

Finds upper A-efficiency bounds for regular block designs.

Usage

A_bound(n, v, b)

Arguments

n the total number of plots in the design.

v the total number of treatments in the design.

b the total number of blocks in the design.
Details

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

References


Examples

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

blocks
Block designs for unstructured treatment sets

Description

Constructs randomized nested blocks for unstructured treatment sets down to any feasible depth of nesting.

Usage

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

Arguments

treatments the required number of treatments partitioned into sets of equally replicated treatments.
replicates the treatment replication numbers for each partitioned treatment set.
blocks the number of nested blocks for each level of nesting from the top 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 with arbitrary depth of nesting for any arbitrary number of unstructured treatment sets.

treatments is a set of numbers partitioning the total number of treatments into sets of equi-replicate treatments.

replicates is a matching set of treatment replication numbers, one for each treatment set in the partition.

blocks is a set of levels where each number is the number of levels nested within each level of the preceding factor. The top single level super-block need not be defined.

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 in 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 by reducing an 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

<table>
<thead>
<tr>
<th>Treatments</th>
<th>A table showing the replication number of each treatment in the design.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Data frame giving the optimized block and treatment design in plot order.</td>
</tr>
<tr>
<td>Plan</td>
<td>Data frame showing a plan view of the treatment design in the bottom level of the design.</td>
</tr>
</tbody>
</table>
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


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.
brightness: 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)
brightness: 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)
brightness: blocks(100,4,list(4,10))

design

General block and treatment designs.

Description

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

```r
design(
  treatments,  # a single treatment factor or a data frame containing any combination of treatment factors
  blocks = NULL,  # a single block factor or a data frame containing one or more qualitative level block factors
  treatments_model = NULL,  # a treatment model formula for the required treatment design.
  weighting = 0.5,  # a weighting factor between 0 and 1 for weighting the 2-factor interaction effects of factorial blocks.
  searches = NULL,  # the maximum number of local optima searched at each stage of an optimization.
  seed = NULL,  # an integer initializing the random number generator.
  jumps = 1  # the number of pairwise random treatment swaps used to escape a local maxima.
)
```

**Arguments**

- **treatments**: a single treatment factor or a data frame containing any combination of treatment factors with any combination of treatment vectors.
- **blocks**: a single block factor or a data frame containing one or more qualitative level block factors in the required order of fitting.
- **treatments_model**: a treatment model formula for the required treatment design.
- **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 a data frame containing one or more qualitative or quantitative level treatment vectors defining a set of candidate treatments. If the size of the candidate set is different from the size of the block design, or if the treatments_model is structured, the treatment design is optimized by selection with replacement, otherwise the full fixed candidate set is used for the treatment design.

blocks is a factor or a data frame containing one or more qualitative level block factors. The design is optimized for each block factor added sequentially from left to right with the optimization constrained to be constant for all previously added block factors. The length of the blocks object defines the total number of plots in the design.

treatments_model is either a single formula or a compound formula split by the | operator. The left hand side of each |, assuming all remaining | are replaced by +, is a partial model formula. Partial model formula define partial design matrices which are fitted and optimized sequentially from left to right. 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 partial model is the generalized variance of the treatment design for that model (D-optimality) and the design efficiency is the ratio of the generalized variance of the full candidate treatment model for the required treatment model relative to the generalized variance of the optimized design. The efficiency is necessarily less than or equal to 1 for purely factorial models but may exceed 1 for polynomial models.
For crossed blocks, a weighting factor \( w \) is used to determine the relative importance of the block main effects versus the block 2-factor interaction effects. If \( w = 0 \) the algorithm fits a simple additive main effects model whereas if \( w = 1 \) the algorithm fits both main effects and 2-factor interaction effects. For intermediate \( 0 < w < 1 \), the 2-factor interaction effects are downweighted relative to the main effects where the smaller the value of \( w \) the greater the downweighting. The default weighting is 0.5 and provided that all block effects are estimable, this weighting gives a compromise design where all main effects and 2-factor interaction effects are included but where the main effects are given greater relative importance than the 2-factor interaction effects. See vignette(package = "blocksdesign") for more details.

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**


**Examples**

```r
## 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 = factor(rep(1:12,4)),blocks)

## 6 replicates of 5 treatments with 2 super blocks of size 15 and 10 nested blocks of size 3
blocks=data.frame(Main = gl(2,15,30), Sub = gl(10,3,30))
design(treatments=gl(5,1,30),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))
design(treatments = factor(1:12),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
blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48), subCols = gl(12,1,48))
design(treatments = factor(1:12),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).
blocks = data.frame(Rows = gl(4,13), Cols = gl(13,1,52))
design(treatments = factor(1:13),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 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=factor(1:48),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=factor(1:48),blocks,searches=5)
design(treatments=factor(1:48),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 = 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] ) ) ) 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"

# 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=5)

## 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)

## 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 in the blocks model as a dual design
## 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


### Usage

data(durban)

### Format

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

---

**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).
isPrime

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

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
isPrimePower

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 \( s = N^{\frac{1}{i}} \), 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 sets of mutually orthogonal Latin squares (MOLS) of dimension \( p^q \) where p is prime. The number of squares r can be any number less than \( p^q \) and the exponent q can be any value from 1 up to a maximum dependent on the choice of p. The permitted values of p and q are:
### MOLS

<table>
<thead>
<tr>
<th>prime p</th>
<th>maximum q</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>11 13 17 19</td>
<td>3</td>
</tr>
<tr>
<td>23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97</td>
<td>2</td>
</tr>
<tr>
<td>101 103 107 etc.</td>
<td>1</td>
</tr>
</tbody>
</table>

**Usage**

\[ \text{MOLS}(p, q, r) \]

**Arguments**

- **p**: is any odd prime greater than one
- **q**: is any suitable integer power (see description for available powers)
- **r**: is any number of squares up to a maximum of \( p^{\ast q} - 1 \)

**Details**

Generates MOLS by cyclic permutation of a basic Latin square \( L_0 \) constructed from a vector of ordered elements \( X \) of a prime-power finite field of size \( p^{\ast q} \) and a unit vector \( 1 \) of length \( p^{\ast q} \) (see Chapter 1 of Raghavarao 1971).

\[ L_0 = \text{crossprod}(X, t(1)) + \text{crosssprod}(1, t(X)) \]

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 for a \( p \ast q \times p \ast q \) square classified by rows and columns with a separate column for the allocation of treatments to each individual square.

**Value**

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

**References**


**Examples**

\[ \text{MOLS}(2, 3, 7) \]
\[ \text{MOLS}(3, 2, 4) \]
\[ \text{MOLS}(3, 3, 4) \]
\[ \text{MOLS}(23, 2, 2) \]
Index

* Topic **data**
  durban, 11

A_bound, 3

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

design, 3, 6
durban, 11

HCF, 11

isPrime, 12
isPrimePower, 13

MOLS, 13