

Vignette for ‘blocksdesign’ package

Summary

Complete replicate block designs are the designs of choice for comparative linear model experiments and are fully efficient for treatment comparisons. However, complete block designs with many treatment plots may not provide good homogeneity of variance and then designs with smaller incomplete nested blocks may be required. A single set of incomplete blocks can provide improved homogeneity of variance for small to medium size experiments but for experiments with a large or very large numbers of treatments, a single set of nested blocks may not be adequate and then multi-level nesting may be required. This vignette discusses the design of experiments with a hierarchy of multi-level blocks nested within the same experiment including multi-level designs for experiments with crossed block factors.

Introduction

Comparative experiments often involve estimation of treatment effects against a background of high non-treatment variability and effective control of background variability is essential for good treatment estimation. The most common method for field trials is the complete randomized blocks design in which every treatment is represented in every block in proportion to its replication. Complete block designs can be effective against a range of nuisance effects such as patchy fertility, row-and-column effects or even the residual effects of previous treatments but for large designs with many treatments, complete randomized blocks may be too large to give good homogeneity of variance. In that situation, incomplete blocks containing fewer than a complete set of treatments in each block can be used to obtain improved homogeneity of variance within blocks.

Incomplete block designs with a single level of nesting assume that heterogeneity of background variation can be partitioned between a single set of nested but for experiments with many treatments, heterogeneity of variation may occur over a range of scales of measurement. In that situation, designs with a single level of nesting may not be adequate to capture all the important sources of non-treatment variability and repeated or multi-level nesting over a range of block sizes may be needed to cover the full range of scales of measurement.

Experiments in agriculture and biology can involve qualitative treatment comparisons such as variety comparisons or can involve quantitative comparisons such as rates of fertilizer application. Choice of the treatment design for a qualitative treatment set is usually pre-determined by the requirements of the experiment but the best choice of design for a quantitative treatment can be complex and may require computer methods for the optimum selection of treatment combinations. Good design of nested incomplete blocks also usually requires computer methods. The ‘blocksdesign’ package is intended to provide an integrated general purpose design package for both treatment and block design, especially for field and crop experiments.

Multi-level nesting

The purpose of multi-level nesting is to provide a range of choices of block structures and block sizes at the analysis stage of an experiment and in this section, we discuss methods for incorporating a range of block structures and sizes at the design stage. In general, block models should be based only on block structures that were built into the design at the design stage, as it is not normally advisable to make *post hoc* searches for block structures that were not envisaged in the original design.

Model

Let \mathbf{t} be a vector of fixed treatment effects for a set of v treatments and let $\mathbf{b}_i, i = 1 \dots u$ be vectors of fixed block effects for u sets of blocks with $k_i, i = 1 \dots u$ sets of levels, respectively. Let \mathbf{T} be a $n \times v$ treatment design indicator matrix and let $\mathbf{B}_i, i = 1 \dots u$ be block design indicator matrices of dimension $n \times k_i$ for $i = 1 \dots u$ respectively. Assuming \mathbf{y} is a vector of observations and \mathbf{e} is a vector of model residuals, a multi-stratum blocks model for an experiment with n observations and u sets of block factors can be written:

$$\mathbf{y} = \mathbf{T}\mathbf{t} + \mathbf{B}_1\mathbf{b}_1 + \dots + \mathbf{B}_u\mathbf{b}_u + \mathbf{e} \quad (1)$$

Design

Let \mathbf{t}_f be a factor with v treatment levels and let $\mathbf{b}_{fi}, i = 1 \dots u$ be factors of block levels for the set of blocks $\mathbf{B}_i, i = 1 \dots u$. Let $\mathbf{T}_f = \mathbf{T}\mathbf{t}_f$ be the actual allocation of treatment factor levels to plots and let $\mathbf{D}_f = (\mathbf{B}_1\mathbf{b}_{f1}, \dots, \mathbf{B}_u\mathbf{b}_{fu})$ be a data frame of the actual allocation of block factor levels to plots. Then the following 'blocksdesign' function gives the actual optimized allocation of treatments to plots:

$$\mathbf{T}_{opt} = \text{design}(\mathbf{T}_f, \mathbf{D}_f)\$Design\$treatments \quad (2)$$

The blocks design algorithm optimizes \mathbf{T}_{opt} by conditionally swapping pairs of rows of \mathbf{T} for each set of blocks in $\mathbf{B}_i, i = 1 \dots u$, taken one at a time, until no further improvement is possible. Conditional swapping means that for any block design matrix $\mathbf{B}_j, j = 2 \dots u$, the treatment swaps for \mathbf{B}_j must be restricted within the levels of all previous blocks, $\mathbf{B}_i, i = 1 \dots j - 1$. Essentially, this means that the blocks of each successive set must be nested within or crossed with the blocks of all previous sets. In particular, the algorithm cannot work if successive blocks are grouping factors for previous blocks in the sequence, which means that the algorithm cannot be used for agglomerating smaller blocks into larger blocks.

For crossed blocks designs, the relative importance of the various factorial block interaction effects must also be considered and the `blocksdesign :: design()` function has a parameter $0 \leq w \leq 1$ that differentially weights the relative importance of the main effects of blocks versus their 2-factor interactions. Provided that the interaction effects are estimable, w weights the relative importance of the 2-factor interaction effects such that $w = 0$ optimizes main effects only, $w = 1$ optimizes main effects and 2-factor interaction effects equally while $0 < w < 1$ gives an intermediate design where the importance of the 2-factor block interaction effects depends on w . See `library('blocksdesign')` and `help(design)` for full details and examples of the use of 'blocksdesign' and for applications of the weighting option. Further details and examples are given in `vignette('blocksdesign')`.

Optimization criterion

The optimization criterion used by 'blocksdesign' is D-optimality which maximizes the determinant of the treatment information matrix or, equivalently, minimizes the determinant of the treatment variance matrix. D-optimality is widely used and has the important property of scale-invariance which means that it can be used for any design including designs with a range of quantitative and qualitative treatments, see Mitchell (1974) and Atkinson et. al. (2007). An alternative criterion is A-optimality, (see John and Williams 1998, Chapter 2) which minimizes the average variance of pairwise treatment differences, assuming equal treatment replication. Some authors, e.g. Jones et. al. (2020), consider that A-optimality is a better criterion for unstructured treatment sets than D-optimality but A-optimality is not currently an option in 'blocksdesign'.

Treatment design

Unstructured treatments

Unstructured treatments have no underlying treatment model and the only meaningful comparisons are the pairwise differences between individual treatments. Treatment design for unstructured treatments is chiefly concerned with the choice of individual treatments and the individual treatment replication and these choices usually depend on the purposes and economics of a trial. Replication need not be equal for all treatments and often it is desirable to increase the replication on certain individual treatments, for example when certain treatments are controls or standards against which the remaining treatments are to be compared. Sometimes, perhaps due to lack of resources or lack of experimental material, some treatments may need to be un-replicated (conventionally they have a single 'replication'). Usually, the choice of replication will be decided by the experimenter on pragmatic grounds and it is important that any good block design algorithm should be able to provide an efficient block design whatever the choice of treatment replication.

Example

The following example shows a simple basic design for a set of 12 unstructured treatments numbered 1 to 12 each with 4 replicates and a single control treatment, numbered 13, with 8 replicates. The example does not specify a block design and the design defaults to four complete randomised blocks each containing treatments 1 – 12 once and treatment 13 twice. Unstructured treatment designs can be constructed either by the blocks() function or by the :design() function. The design() function is more general than the blocks() function and provides more control over the blocks and treatment design but requires a more detailed set of input parameters. In this example, the blocks() function will be used to provide a simple basic block design:

Model

```
blocks(treatments = c(12,1), replicates = c(4,8))
```

Output

```
$Blocks_model
  Level Blocks D-Efficiency A-Efficiency A-Bound
1 Level_0      4           1           1       1
```

The output shows the block efficiency table for the design where the column labelled 'Level' indicates the depth of nesting. In this case, there is just one set of complete randomized blocks and the level of nesting is zero. The A-efficiency bound is only available for equi-replicate designs with equal block sizes or for complete orthogonal block designs, as in the above example.

The other design outputs for the `blocks()` function include `$Treatments`, showing a table of the treatment replicates, `$Design` showing a table of the allocation of treatments to blocks and `$Plan` showing a plan of the allocation of treatments to blocks in the bottom level of blocking.

Structured treatments

Structured treatments have an underlying model such as a response surface for quantitative level factors or a factorial model for qualitative level factors. Response surfaces and factorial designs assume an empirical linear model for treatment effects and efficient design usually requires the optimization of a design criterion derived from the information matrix of the linear design matrix. The most general design criterion is D-optimality (see Atkinson et. al. 2007), which maximizes the determinant of the design information matrix and D-optimality is the criterion used by 'blocksdesign' for the numerical optimization of all general, non-orthogonal, treatment designs.

The design algorithm used by 'blocksdesign' selects an optimal set of treatment combinations from a candidate set of treatments. The candidate set contains all the feasible treatment combinations that might occur in the final optimized design and the design algorithm selects those combinations that optimize the treatment information matrix. The treatments are selected from the candidate set by swapping which means that the final selected design can contain, at most, only a single copy of any particular plot in the candidate set. The candidate set must therefore be at least as large as the required treatment design and if a treatment set is supplied that is smaller than the required design, 'blocksdesign' will replicate the set a sufficient number of times to cover the required size of design. If the candidate set contains replicates of individual treatments then the final optimized design can replicate each treatment up to the number of replicates of that treatment in the candidate set. The optimization of designs with quantitative level treatment factors may depend on the available number of replicates in the candidate set so exploration of the effects of different sizes of candidate sets is recommended.

The treatments model vector is a character vector containing one or more nested treatments formula where the treatment factors are optimized for each formula taken in order but with all previously optimized treatment factors held constant. Sequential model fitting can provide flexibility for fitting factors or variables of different status or importance.

For example, the following example constructs a treatment design for 4 varieties with 3 levels of N and 3 levels of K assuming a degree-2 fertilizer response surface and a design with two blocks each of size 12.

Model 1

```
treatments = expand.grid(Variety = factor(1:4), N = 1:3, K = 1:3)
blocks = data.frame(main = gl(2,12))
model = " ~ (Variety + N + K)^2 + I(N^2) + I(K^2) "
design(treatments, blocks, treatments_model = model, searches=10)
```

Output (model 1)

```
$Design
  main Variety N K
1     1      1 2 3
2     1      4 3 3
3     1      3 3 3
4     1      1 3 1
5     1      2 1 3
6     1      2 3 1
7     1      1 1 3
8     1      3 1 1
9     1      2 2 2
10    1      4 1 1
11    1      3 1 2
12    1      3 3 2
13    2      4 3 1
14    2      1 1 1
15    2      2 1 1
16    2      4 1 3
17    2      1 3 3
18    2      2 3 3
19    2      3 3 1
20    2      1 3 2
21    2      4 2 2
22    2      3 2 1
23    2      3 2 3
24    2      3 1 3

$Treatments_model
      Treatment.model Model.DF D.Efficiency
1 ~ (Variety + N + K)^2 + I(N^2) + I(K^2)      14      1.052045
```

Model 1 has a complete candidate set of the 48 factorial combinations of Varieties with N and K and optimizes all terms of the treatment model simultaneously resulting in a treatment design with between 5 and 8 combinations of each of the four variety levels. Most research workers would expect equal replication on each variety level and equality of the numbers of combinations of the four variety levels can be forced by fitting the variety factor first and then fitting the rest of the model with the variety effects held constant. Model 2 shows how this can be achieved by first fitting just the Variety model and then fitting the full combined treatment model for Variety and fertilizer effects conditional on the fixed Variety effects.

Model 2

```
treatments = expand.grid(Variety = factor(1:4), N = 1:3, K = 1:3)
blocks = data.frame(main = gl(2,12))
model = c("~Variety", "~Variety + (Variety + N + K)^2 + I(N^2) + I(K^2)")
design(treatments, blocks, treatments_model = model, searches=10)
```

```
$Design
  main Variety N K
1     1         4 1 1
2     1         3 1 1
3     1         4 1 3
4     1         1 2 1
5     1         2 2 1
6     1         4 3 2
7     1         3 1 2
8     1         1 3 3
9     1         3 3 3
10    1         2 1 3
11    1         2 3 1
12    1         1 1 3
13    2         2 3 3
14    2         3 3 2
15    2         1 1 1
16    2         2 2 3
17    2         2 1 1
18    2         4 3 3
19    2         1 2 3
20    2         3 3 1
21    2         4 1 2
22    2         1 3 1
23    2         3 1 3
24    2         4 3 1
```

```
$Treatments_model
Treatment.model Model.DF D.Efficiency
1 ~ Variety 3 1
2 ~ Variety + (Variety + N + K)^2 + I(N^2) + I(K^2) 14 1.043662
```

The overall D-efficiency factor of Model 2 is slightly less than the overall D-efficiency of Model 1 but the model now has an equal division of the 24 factor combinations into 6 combination for each level of variety which most research workers would regard as a necessary requirement for a good design.

NB the ^2 squaring operator is an interaction operator unless 'protected' by the I() operator.

Simple nested block designs

In many situations, comparability of treatments can be improved by grouping experimental units into blocks. Blocks should be as homogeneous as possible and the choice of blocks design can be critical for the success of an experiment. The most basic type of block design is the randomized complete blocks design where each block contains one or more complete replicate sets of treatments. Complete

blocks estimate all treatment effects fully within blocks and are usually the best choice for small experiments. However, for large experiments, variability within complete blocks can be large and then it may be beneficial to further sub-divide complete replicate block into smaller nested sub-blocks to improve the precision of comparisons within-sub-blocks.

Multi-level nesting

Complete replicate blocks with a single level of nesting are resolvable incomplete block designs and are widely used in practical research. Treatment information is estimated both within and between blocks and a fully informative analysis requires the combination of block treatment information using some form of mixed-model analysis, Piepho and Edmondson (2018). The aim of good block design is to maximize the precision of estimation of treatment effects and for a single level of nesting block designs can be optimized by maximizing the information content of the incomplete blocks design. Various design criteria have been considered for block designs (John & Williams 1998) but the most general design criterion is D-optimality. The D-optimality criterion maximizes the determinant of the design information matrix and is the criterion of choice used by the 'blocksdesign' algorithm.

Although resolvable block designs with a single level of nesting work well for small or moderate numbers of experimental units, a single level of nesting may be inadequate for large experiments such as field variety trials which may involve scores or hundreds of treatments. Multi-level nesting gives a series of nested blocks where the nested inter-block space at each level of nesting can be assumed to have good homogeneity of variance. A mixed model analysis of a multi-level nested block design using modern mixed model design is straightforward and allows the proper weighted combination of treatment information from each inter-block space.

The blocks() function of 'blocksdesign' is a special recursive function for simple multi-level nested block designs for unstructured treatment sets. The function generates designs for treatments with arbitrary levels of replication and arbitrary depth of nesting and each successive set of blocks is optimized within the levels of each preceding set of blocks using conditional D-optimality. 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. See John and Williams (1998) for a definition of A-efficiency.

The following examples show a simple recursively nested block design for four replicates of 100 treatments with four complete main blocks for the top level of blocks and two levels of nesting where the first level of nesting has 10 sub-blocks of size 10 nested within each main block and the second level of nesting has two sub-sub-blocks of size 5 nested within each nested sub-block.

Model

```
blocks(100, 4, list(4,10,2))$Blocks_model
```

Block efficiencies

	Level	Blocks	D-Efficiency	A-Efficiency	A-Bound
1	Level_0	4	1	1	1
2	Level_1	40	0.90067	0.89189	0.89189
3	Level_2	80	0.78476	0.75946	0.76327

The A-efficiency of the Level_1 blocks is optimal because the Level_1 blocks are lattice blocks based on a pair of orthogonal 10 x 10 Latin squares. The A-efficiency of the Level_2 blocks is close to the theoretical upper A-bound which shows that the constraints of multi-level nesting have not significantly reduced the efficiency of the Level_2 blocks design.

Factorial nested block designs

Sometimes it can be advantageous to use a fully crossed factorial blocks design in field trials. For example, factorial row-and-column blocks are sometimes used to accommodate physical row and column effects in a field layout. Factorial block designs are often assumed to fit a simple additive main effects model but additivity of main effects is a very strong assumption and may not be fully valid for blocks with many crossed levels. For that reason, the 'blocksdesign' algorithm can fit block main effects and block 2-factor interaction effects weighted by an assumed model for the relative importance of main effects versus 2-factor interaction effects, assuming that all model effects are estimable.

Weighted factorial treatment models

Let \mathbf{T} be a matrix of contrasts for a set of treatments factors and let \mathbf{B}_1 and \mathbf{B}_2 be contrast matrices for the additive effects of two sets of crossed block factors. Let $\mathbf{B}_{2:1}$ be the matrix of two-factor interactions between \mathbf{B}_1 and \mathbf{B}_2 . Then the full block and treatment design matrix is:

$$\mathbf{T} + \mathbf{B}_1 + \mathbf{B}_2 + \mathbf{B}_{2:1}$$

Assume $\mathbf{B} = (\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_{2:1})$ is of full rank and assume a singular value decomposition $\mathbf{B} = \mathbf{Q}\mathbf{R}$ where $\mathbf{Q} = (\mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_{2:1})$ is a conformal orthogonal basis for \mathbf{B} . Since $\mathbf{Q}'\mathbf{Q} = \mathbf{I}$ and $(\mathbf{R}'\mathbf{R})^{-1} = \mathbf{R}^{-1}\mathbf{R}'^{-1}$, the block adjusted treatment information matrix $\mathbf{T}'(\mathbf{I} - \mathbf{B}(\mathbf{B}'\mathbf{B})^{-1}\mathbf{B}')\mathbf{T}$ can be re-written as:

$$\mathbf{T}'(\mathbf{I} - \mathbf{Q}_1\mathbf{Q}_1' - \mathbf{Q}_2\mathbf{Q}_2' - \mathbf{Q}_{2:1}\mathbf{Q}_{2:1}')\mathbf{T}$$

The 'blocksdesign' algorithm optimizes a weighted treatment information matrix:

$$\mathbf{T}'(\mathbf{I} - \mathbf{Q}_1\mathbf{Q}_1' - \mathbf{Q}_2\mathbf{Q}_2' - w^2 \times \mathbf{Q}_{2:1}\mathbf{Q}_{2:1}')\mathbf{T}$$

where the two-factor interaction term $\mathbf{Q}_{2:1}\mathbf{Q}_{2:1}'$ is down-weighted by an arbitrary scalar w^2 for $0 \leq w \leq 1$.

For $w = 0$, the weighted information matrix gives the usual additive crossed blocks model whereas if $w = 1$, the weighted information matrix gives a full factorial blocks model. Intermediate values of w , down-weight the block interaction effects according to the square of w . The best choice of w will be unknown, but the effects of different choices on the attained efficiency factors of the various factorial block effects can be found by trial error at the design stage.

The following three examples show crossed blocks design for 4 replicates of 12 treatments with 4 main rows and 4 main columns. and blocks of size 3 nested within each row-by-column intersection. The row blocks are added first and will always comprise complete replicate blocks. The column blocks are added after the row blocks and the efficiency of the column block main effects and the

row-by-column interaction block effects will depend on the choice of weighting. The three examples show the effects of three different choices of weighting parameter on the relative importance of the 2-factor row-by-column interaction effects.

In Model 1 the weighting is zero and additive column block effects are orthogonal and are estimated with full efficiency but the rows-by-columns block interaction effects are estimated with low efficiency.

In Model 2 the weighting is 0.5 and the rows-by-columns block effects are estimated with improved efficiency relative to Model 1 while the additive column block effects still remain orthogonal and are still estimated with full efficiency.

In Model 3 the weighting is 1 and the rows-by-columns block interaction effects are estimated with the maximum possible efficiency. However, the additive column blocks are no longer orthogonal and are estimated with efficiency less than one. The Model 3 design is equivalent to an incomplete block design with four complete main blocks (rows) and with four sub-blocks (row-by-column interaction blocks) nested within each main block.

1) Weighting = 0 giving an additive main effects design for rows and columns

Model 1

```
treatments = factor(1:12)
blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48))
design(treatments,blocks,searches=200,weighting=0)$Blocks_model
```

Output

	Blocks	Levels	D_Effic	A_Effic	Interactions	Int_levs	Int_D_Effic	Int_A_Effic
1	Rows	4	1	1	Rows	4	1.000000	1.000000
2	Cols	4	1	1	Rows*Cols	16	0.682796	0.6316198

2) Weighting = 0.5 giving a compromise design for rows, columns and rows:columns blocks

Model 2

```
treatments = factor(1:12)
blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48))
design(treatments,blocks,searches=200,weighting=0.5)$Blocks_model
```

Output

	Blocks	Levels	D_Effic	A_Effic	Interactions	Int_levs	Int_D_Effic	Int_A_Effic
1	Rows	4	1	1	Rows	4	1.000000	1.000000
2	Cols	4	1	1	Rows*Cols	16	0.7176709	0.7096774

3) Weighting = 1 giving a fully crossed rows-by-columns design

Model 3

```
treatments = factor(1:12)
blocks = data.frame(Rows = gl(4,12), Cols = gl(4,3,48))
design(treatments,blocks,searches=200,weighting=1)$Blocks_model
```

Output

	Blocks	Levels	D_Effic	A_Effic	Interactions	Int_levs	Int_D_Effic	Int_A_Effic
1	Rows	4	1.0000000	1.0000000	Rows	4	1.0000000	1.0000000
2	Cols	4	0.9144364	0.9034965	Rows*Cols	16	0.7176709	0.7096774

In this example, Model 2 gives a fully orthogonal set of main column blocks and fully efficient rows-by-columns design compared with Model 3 which has full weighting for row-by-column effects. Usually, for crossed blocks designs, not all rows, columns and rows-by-columns can be optimized simultaneously but this example is a special design called a Trojan square (Edmondson 1998) which has the property that the main rows and columns blocks design and the rows-by-columns interaction design can all be optimized simultaneously. This exemplifies the utility of the weighting method for designs with estimable crossed blocks interaction effects. In the general case, trial and error methods can be used to find a good choice of weighting that gives a good compromise design with good efficiencies on all the required block structures.

MOLS and lattice block designs

Mutually orthogonal Latin squares (MOLS) can be used for constructing a range of square and rectangular lattice block designs, (Cochran and Cox. 1992). Square lattice designs have v^2 equally replicated treatments in v sets of incomplete blocks of size v nested within complete replicate blocks. Rectangular lattice designs have $v(v-1)$ equally replicated treatments in v sets of incomplete blocks of size $v-1$ nested within complete replicate blocks. Square lattice designs for r replicates require $r-2$ MOLS whereas rectangular lattice designs require $r-1$. The MOLS function will construct a complete set of v^2-1 orthogonal Latin squares of size $v \times v$ whenever v is a prime or a prime power whereas the GraecoLatin function will construct a pair of MOLS for any odd v or any even $v < 30$ (except 6).

Prime power MOLS

The MOLS function has three parameters, p , q and r , and constructs r sets of mutually orthogonal Latin squares (MOLS) of dimension $p \times q$ for prime p and integer power q where $r < p \times q$. Memory issues mean that the maximum size of the exponent q for specific p is restricted to the values shown in the table below: For $r = 1$, the function constructs Latin squares of any size for any p and any q .

Table 1 Primes p and upper bounds q for MOLS of size p^q

P	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

The MOLS are generated 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 this field 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 $(p^q) \times (r+2)$ with a column for the rows classification and the columns classification of the design and a treatments classification for each MOL in the required set of r replicates.

GraecoLatin MOLS

The GraecoLatin function has a single parameter N for the required number of treatments and constructs pairs of 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 Table 1
- iii) any even valued $N \leq 30$ except for 6 or 2

Even valued, non-prime power Graeco-Latin squares for $N \leq 30$ are generated by the methods given by Street & Street (1987), Chapters 6 and 7.

The output is a single data frame of size $(p^q) \times (r+2)$ with a column for the rows classification and a column for the columns classification of the design and a column for each treatment classification of each square of the required set of MOLS.

Lattice designs

The 'blockdesign' package constructs general designs by a random swapping algorithm except in the special case of square or rectangular lattice designs. Square lattices have v^2 treatments and blocks of size v where the number of replicates r must be not greater than the available number of MOLS plus 2. Rectangular lattice designs have $v(v-1)$ treatments in blocks of size v where the number of replicates r must be not greater than the available number of MOLS plus 1. These designs are constructed algebraically by the methods given in Chapter 10 of Cochran and Cox (1992) and will always attain the theoretical efficiency bound whenever a suitable set of MOLS is available.

Tables 2 and 3 show the theoretical maximum number of replicates for any square lattice with N not greater than 400 or any rectangular lattice with N not more than 380. Any square or rectangular lattice constructed by 'blocksdesign' for a design with a suitable number of MOLs will attain the theoretical A-efficiency upper bound provided that the number of replicates does not exceed the theoretical bound. Larger designs can be found using larger MOLS if required.

Table 2 Available square lattices up to N = 400 and the maximum available number of replicates for each design

v	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
N	9	16	25	36	49	64	81	100	121	144	169	196	225	256	289	324	361	400
r	4	5	6	3	8	9	10	4	12	4	14	4	4	17	18	4	20	4

Table 3 Available rectangular lattices up to N = 380 and the maximum available number of replicates for each design

v	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
N	6	12	20	30	42	56	72	90	110	132	156	182	210	240	272	306	342	380
r	3	4	5	2	7	8	9	3	11	3	13	3	3	16	17	3	19	3

In practice, the range of available lattice designs is highly restrictive and general block designs constructed by numerical optimization will normally be required for real experiments.

Some additional examples

Durban et.al. (2003) discussed an experiment with two replicates of 272 spring barley varieties arranged in an array of 34 columns (east-west) and 16 rows (north-south) subject to the constraint that rows 1-8 contained one complete set of treatment replicates and rows 9-16 contained the other. They showed that an analysis based on a conventional additive row-and-column model was inadequate due to residual trends within rows. Under these circumstances, it seems natural to model rows and columns using nested column blocks.

The original barley variety trial was designed as a simple row-and-column design with 16 rows and 34 columns but, for the purposes of this example, it is reasonable to assume a set of nested column blocks imposed on the original design and to assume interacting row-and-column blocks.

Model 1

The original design had 16 rows and 34 columns and these are modelled here by a row factor Rows with 16 levels crossed with three nested columns factors Col1, Col2 and Col3 with 4, 8 and 34 levels, respectively. Assuming 34 plots within each row, not all the nested column block factors can be of equal width and two column blocks from each of Col1, Col2 and Col3 must be an extra plot wide.

Input

```
treatments = factor(rep(1:272,2))
Reps = factor(rep(1:2,each=272))
Rows = factor(rep(1:16,each=34))
Col1 = factor(rep(rep(1:4,c(9,8,8,9)),16))
Col2 = factor(rep(rep(1:8,c(5,4,4,4,4,4,4,5)),16))
Col3 = factor(rep(1:34,16))
blocks = data.frame(Reps,Rows,Col1,Col2,Col3)
design(treatments, blocks, searches=1)
```

Output

First-order model effects

First_order_model effects		D.Efficiency	A.Efficiency
(Reps)	1	1.00000	1.00000
(Reps+Rows)	15	0.96492	0.95091
(Reps+Rows+Col1)	18	0.96060	0.94593
(Reps+Rows+Col1+Col2)	22	0.95084	0.93288
(Reps+Rows+Col1+Col2+Col3)	48	0.88785	0.84983

Second-order model effects

Second_order_model effects		D.Efficiency	A.Efficiency
(Reps)^2	1	1.00000	1.00000
(Reps+Rows)^2	15	0.96492	0.95091
(Reps+Rows+Col1)^2	63	0.84423	0.78183
(Reps+Rows+Col1+Col2)^2	127	0.67750	0.54116
(Reps+Rows+Col1+Col2+Col3)^2	543	0.00000	0.00000

Model 2

The original design had 34 plots in each row and not all columns in each set of columns were of equal width. A simpler column blocks design can be based on a design with 32 plots in each row but this design requires 17 rows to accommodate the two replicates of 272 treatments. The following design shows a 'Reps' factor with 3-levels, which splits the design into rows 1:8 with 256 plots, row 9 with 32 plots and rows 10:17 with 256 plots. The design algorithm allocates 256 treatments, all different, to rows 1:8, 32 treatments, all different, to row 9 and 256 treatments, all different, to rows 10:17.

Input

```
treatments = factor(rep(1:272,2))
Reps = factor(c(rep(1,256),rep(2,32),rep(3,256)))
Rows = factor(rep(1:17,each=32))
Col1 = factor(rep(rep(1:4,each=8,17)))
Col2 = factor(rep(rep(1:8,each=4,17)))
Col3 = factor(rep(1:32,17))
blocks = data.frame(Reps,Rows,Col1,Col2,Col3)
d=design(treatments, blocks, searches=1)
# incidences to check the treatment partition between the three
levels of Reps
table(d$Design[,1],d$Design[,6])
```

Output

	First_order_model	effects	D.Efficiency	A.Efficiency
1	(Reps)	2	0.99756	0.99665
2	(Reps+Rows)	16	0.96256	0.94786
3	(Reps+Rows+Col1)	19	0.95825	0.94290
4	(Reps+Rows+Col1+Col2)	23	0.94852	0.92994
5	(Reps+Rows+Col1+Col2+Col3)	47	0.89042	0.85338

	Second_order_model	effects	D.Efficiency	A.Efficiency
1	(Reps)^2	2	0.99756	0.99665
2	Reps+Rows)^2	16	0.96256	0.94786
3	(Reps+Rows+Col1)^2	67	0.83372	0.76597
4	(Reps+Rows+Col1+Col2)^2	135	0.65676	0.51247
5	(Reps+Rows+Col1+Col2+Col3)^2	543	0.00000	0.00000

Row 9 of Model will be split between the two replicate sets with 8 plots from row 9 combined with rows 1:8 to give replicate 1 and eight plots from row 9 combined with rows 10:17 to give replicate 2. Overall, Model 2 should give a smaller maximum pairwise SED for all possible treatment comparisons than Model 1.

References

- Atkinson, A.C, Donev, A.N. & Tobias, R. D. (2007). Optimum Experimental Designs, with SAS. Oxford, Oxford University Press.
- 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.
- Cochran. W. G., and Cox. G. M. (1992) Experimental Designs, 2nd Edition. Wiley.
- 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.
- Edmondson R. N. (1993). Systematic Row-and-column Designs Balanced for Low Order Polynomial Interactions between Rows and Columns. J. R. Statist. Soc. B (1993) 55, No. 3, pp. 707-723
- Edmondson, R. N. (1998). Trojan square and incomplete Trojan square designs for crop research. Journal of Agricultural Science, Cambridge (1998), 131, 135–142.
- Edmondson, R. N. (2020). Multi-level Block Designs for Comparative Experiments. JABES. <https://link.springer.com/article/10.1007/s13253-020-00416-0>
- Hansen, T. & Mullen, G. L. (1992) Primitive polynomials over finite fields, Mathematics of Computation, 59, 639-643 and Supplement.

John, J. A & Williams, E. R. (1998). Cyclic and Computer Generated Designs. 2nd Edition, Chapman and Hall.

Piepho, Hans-Peter & Edmondson R. N. (2018). A tutorial on the statistical analysis of factorial experiments with qualitative and quantitative treatment factor levels. [Journal of Agronomy and Crop Science](#), 204, 429-455.

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

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