Vignette for 'blocksdesign' package

Summary

Randomized complete blocks are the designs of choice for small or medium sized experiments. For large experiments, however, a randomized complete blocks design may contain too many plots to give good control of plot-to-plot variability and then a resolvable incomplete blocks design with a single set of incomplete blocks nested within main blocks may be a better choice. Incomplete block designs with a single level of nesting can provide improved designs for medium size experiments but for large experiments with many plots, multi-level nesting is required to provide the range of block sizes needed to control variability over a range of scales of measurement. This vignette discusses an R software package for multi-level nesting in general block designs.

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 designs for field trials are randomized complete block designs in which every treatment is represented in every block in proportion to its replication. Randomized complete block designs can be very 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 control of non-treatment variability. In that situation, incomplete blocks containing fewer than a complete set of treatments in each block may give improved control of non-treatment variability.

Incomplete block designs with a single level of nesting assume that heterogeneous background variation can be partitioned between a single set of nested blocks. For large experiments with many treatments, however, heterogeneity of variability may occur over a wide range of scales of measurement and then designs with a single level of nesting may not be adequate. In that situation, repeated or multi-level nesting may be needed to capture all the non-treatment sources of variability.

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 qualitative treatments is usually pre-determined by the requirements of the experiment but the best choice of design for quantitative treatments can be complex and may require computer methods for optimum treatment design. Good design of nested incomplete blocks also requires computer methods and the 'blocksdesign' package provides 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. 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 at the design stage. In this section, we discuss methods for incorporating a range of block structures and sizes at the design stage of an experiment.

Design

Let $\mathbf{T}_f$ be a treatment factor allocating treatment factor levels to plots and let $\mathbf{B}_1$, …,$\mathbf{B}_\mathrm{u}$ be a set of u block factors allocating block levels to plots. Let $\mathbf{D}_\mathrm{f} = (\mathbf{B}_1, \cdots, \mathbf{B}_\mathrm{u})$ be a data frame containing the nested block factors in decreasing order of nesting. Then the allocation of treatments to plots can be optimized by the 'blocksdesign' function: $design(\mathbf{T}_f, \mathbf{D}_f)$.

The design algorithm optimizes the allocation of treatments to blocks by conditionally swapping pairs of rows of $\mathbf{T}_f$ for each set of blocks in $\mathbf{B}_\mathrm{i}, \mathrm{i} = 1 \ldots u$, taken one at a time, until no further improvement is possible. Conditional swapping means that improving swaps for each nested block factor $\mathbf{B}_\mathrm{j}, \mathrm{j} = 2 \ldots u$, are restricted within the levels of all previous blocks, $\mathbf{B}_\mathrm{i}, \mathrm{i} = 1 \ldots j-1$. 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 down-weights the relative importance of 2-factor interactions versus factorial main effects. Provided that the interaction effects are estimable, w weights the relative importance of the 2-factor interaction effects such that $w = 0$ gives main effects optimization only, $w = 1$ gives main effects and 2-factor interaction effects optimization equally while $0 < w < 1$ gives an intermediate design where the information on the 2-factor block interaction effects is down-weighted according to the square of w. See *library('blocksdesign')* and *help(design)* for full details and examples of the use of 'blocksdesign' and for applications of the weighting option.

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 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, some treatments are 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 arranged in four randomized complete blocks. Unstructured treatment designs with equal or near-equal block sizes can be constructed by the blocks() function as follows:

****
Example 1
****

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

Output

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

The design() function is more general than the blocks() function and provides more control over the choice of block sizes and the choice of treatment design but requires a more detailed set of input parameters. For the previous example, the design() function can be used as follows:

****
Example 2
****

```
design(treatments = factor(c(1:12,13,13)),
blocks = factor(rep(1:4,each = 14)))
```

Output

```
$Blocks_model
  Level Blocks D-Efficiency A-Efficiency A-Bound
1     1      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 with a single set of blocks. The A-efficiency bound is only available for equi-replicate designs with equal block sizes.

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 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 structured, non-orthogonal, treatment designs.

The treatment design algorithm 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 without replacement, which means that the final selected design can contain, at most, only a single copy of any particular candidate in the candidate set. The candidate set must be at least as large as the required treatment design and if a treatment set is supplied that is smaller than the required design, the algorithm will replicate the supplied 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 is a character vector containing one or more nested treatments formula, where each nested formula taken in order must contain at least one additional treatment factor. The treatments model is optimized sequentially for each formula with the additional treatment factors in each formula optimized while all previous treatment factors are held constant. Sequential treatment model fitting can provide flexibility for fitting factors or variables of different status or importance.

The following example shows two alternative treatment designs 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.

Example 3 shows a design for the full treatment model fitted simultaneously using a single treatment model formula. This design gives a very unequal division of the 24 plots between the four variety levels and, in this particular realization, variety 1 has 6 plots, variety 2 has 5 plots, variety 3 has 8 plots and variety 4 has 5 plots.

****
Example 3
****

```
treatments = list(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

```
$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


$Blocks_model
     Level Blocks D.Efficiency A.Efficiency Bound
main     1      2    0.9940052    0.9937659     1
```

****

Example 4 shows the same design fitted to the same basic treatment model but this time using a treatment model formula with sequential terms where each term is fitted sequentially and conditional on all previously fitted terms. In this example, the first part of the formula fits the Variety model alone and the second part then fits the full combined treatment model for Variety and fertilizer effects. conditional on the fitted Variety effects. Example 4 is slightly less efficient overall than Example 3

but gives 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.

****

Example 4

****

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

Output

```
$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

$Blocks_model
     Level Blocks D.Efficiency A.Efficiency Bound
main     1      2    0.9916222    0.9911504     1
```

****

Multi-level nesting

Complete replicate blocks with a single level of nesting are widely used in practical research. Treatment information is estimated both within and between blocks and a full analysis requires the combination of block treatment information using a 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 not be adequate for large experiments such as field variety trials which may involve scores or hundreds of treatments. Multi-level nesting gives a mixed model in which the block variability is allowed to change with block size. Mixed model analysis is straightforward using modern software such as lme4 and allows a proper weighted combination of treatment information from each block size.

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. See John and Williams (1998) for a definition of A-efficiency.

Example 5, show a nested blocks design for four replicates of 100 treatments with four complete replicate main 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 sub-block.

```
**********************************************************************************
Example 5
**********************************************************************************

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

Output

```
$Blocks_model
  Level Blocks D-Efficiency A-Efficiency   A-Bound
1     1      4    1.0000000    1.0000000 1.0000000
2     2     40    0.9006742    0.8918919 0.8918919
3     3     80    0.7847727    0.7594912 0.7632672

**********************************************************************************
```

In Example 5, the A-efficiency of the sub-blocks is optimal because the sub-blocks are lattice blocks based on a pair of orthogonal 10 x 10 Latin squares. The A-efficiency of the sub-sub-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 sub-sub-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 all 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 $\mathbf{B} = \mathbf{QR}$ where $\mathbf{Q} = (\mathbf{Q_1}, \mathbf{Q_2}, \mathbf{Q_{2:1}})$ is a conformal orthogonal basis for $\mathbf{B}$. Since $\mathbf{Q'Q} = \mathbf{I}$ and $(\mathbf{R'R})^{-1} = \mathbf{R^{-1}R'^{-1}}$, the block adjusted treatment information matrix $\mathbf{T'}\left(\mathbf{I} - \mathbf{B}(\mathbf{B'B})^{-1}\mathbf{B'}\right)\mathbf{T}$ 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:

$$\mathbf{T'}\left(\mathbf{I} - \mathbf{Q_1}\mathbf{Q_1}' - \mathbf{Q_2}\mathbf{Q_2}' - w^2 \times \mathbf{Q_{2:1}}\mathbf{Q_{2:1}}'\right)\mathbf{T}$$

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.

Example 6-8, show crossed blocks design for 4 replicates of 12 treatments with 4 main rows and 4 main columns with 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.

****

Example 6

****

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

Output

```
$Blocks_model
  First_order effects D.Effic A.Effic  Second_order effects D.Effic A.Effic
1      (Rows)       3       1       1      (Rows)^2       3      NA      NA
2 (Rows+Cols)       6       1       1 (Rows+Cols)^2      15      NA      NA
```

****

In Example 6, 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 have low efficiency (singular in this example).

****

Example 7

****

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

Output

```
$Blocks_model
First_order effects D.Effic A.Effic  Second_order effects  D.Effic  A.Effic
     (Rows)       3       1       1      (Rows)^2       3       NA     NA
(Rows+Cols)       6       1       1 (Rows+Cols)^2      15 0.717671 0.709677
```

****

In Example 7, the weighting is 0.5 and the rows-by-columns block effects are estimated with improved efficiency relative to Example 6 while the additive column block effects remain orthogonal.

```
*******************************************************************************
```
Example 8
```
*******************************************************************************

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

Output

```
$Blocks_model
First_order effects D.Effic A.Effic  Second_order effects  D.Effic  A.Effic
    (Rows)        3 1.000000 1.00000       (Rows)^2        3       NA       NA
(Rows+Cols)       6 0.880561 0.87041  (Rows+Cols)^2       15 0.717671 0.709677

*******************************************************************************
```

In Example 8, the weighting is 1 and the rows-by-columns block interaction effects are estimated with the best possible efficiency. However, the additive column blocks are no longer orthogonal so there will be a loss of efficiency on the estimation of the block main effects.

Note that Example 7 gives a fully orthogonal set of main column blocks and fully efficient rows-by-columns design compared with Example 8 which has full weighting on the rows-by-columns interaction effects.

Usually, not all rows, columns and rows-by-columns effects can be optimized simultaneously for a crossed blocks design 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.

Response surface designs

Quantitative level factors such as rates of application of a fertilizer are often combined with qualitative level factors such as type of fertilizer or variety of treated crop, as discussed in the previous examples, but sometimes designs contain just quantitative level factors all with the same number of levels and then simple response surface designs with special properties of balance can be useful.

The following examples show how 'blocksdesign' can be used to construct efficient response surface designs for three, 3-level quantitative factors assuming a second-order (quadratic) response surface model. A full 3^3 factorial design has 27 points arranged as a $3 \times 3 \times 3$ cube with 8 corner points, 12 edge centre points, 6 face centre points and 1 centre point and a plausible quadratic response surface design could be based on the 21 support points comprising the 8 corner points, the 12 edge centre points and the centre point. Example 9 shows the D-optimal design found by 'blocksdesign assuming 21 support points.

****
Example 9
****

```
treatments = list(V1 = 1:3, V2 = 1:3, V3 = rep(1:3,2))
blocks = data.frame(main = gl(1,21))
model =  ~ (V1 + V2 + V3)^2 + I(V1^2) + I(V2^2) + I(V3^2)
design(treatments,blocks,treatments_model=model,searches=20)
```

Output

```
$Replication
   V1 V2 V3 freq
1   1  1  1    1
2   1  1  2    1
3   1  1  3    1
4   1  2  1    1
5   1  2  3    1
6   1  3  1    1
7   1  3  2    1
8   1  3  3    1
9   2  1  1    1
10  2  1  3    1
11  2  2  2    1
12  2  3  1    1
13  2  3  3    1
14  3  1  1    1
15  3  1  2    1
16  3  1  3    1
17  3  2  1    1
18  3  2  3    1
19  3  3  1    1
20  3  3  2    1
21  3  3  3    1

$Treatments_model
                                     Treatment.model Model.DF D.Efficiency
1 ~(V1 + V2 + V3)^2 + I(V1^2) + I(V2^2) + I(V3^2)         9     1.055267
```

****

The treatments candidate set is of double length which allows any support point to be included in the design twice but, in this example, the overall optimum D-optimum design includes each of the 21 support points once only. The optimum support points do, indeed, comprise the 8 corner points, the 12 edge centre points and the centre point.

The example can be taken further by assuming an optimum design based on 30 support points where each corner point and the centre point could be replicated twice. Example 10 shows the D-optimal design found by 'blocksdesign assuming 30 support points.

```
********************************************************************************
```
Example 10
```
********************************************************************************

treatments = list( V1 = 1:3, V2 = 1:3, V3 = rep(1:3,2))
blocks = data.frame(main = gl(1,30))
model =  ~ (V1 + V2 + V3)^2 + I(V1^2) + I(V2^2) + I(V3^2)
design(treatments,blocks,treatments_model=model,searches=20)


$Replication
   V1 V2 V3 freq
1   1  1  1    2
2   1  1  2    1
3   1  1  3    2
4   1  2  1    1
5   1  2  3    1
6   1  3  1    2
7   1  3  2    1
8   1  3  3    2
9   2  1  1    1
10  2  1  3    1
11  2  2  2    2
12  2  3  1    1
13  2  3  3    1
14  3  1  1    2
15  3  1  2    1
16  3  1  3    2
17  3  2  1    1
18  3  2  3    1
19  3  3  1    2
20  3  3  2    1
21  3  3  3    2

$Treatments_model

                                    Treatment.model Model.DF D.Efficiency

1 ~(V1 + V2 + V3)^2 + I(V1^2) + I(V2^2) + I(V3^2)        9     1.072603

********************************************************************************
```

Example 10 does, indeed, show double replication on each of the eight corner points and double replication on the centre point accounting for all 30 support points. Presumably, the corner points and the centre point are so important for the design that reinforcing these points gives a better improvement in efficiency than would the inclusion of any other design points.

Practical designs for real experiments are likely to be more complex and further examination of designs with 4 or 5-levels which allow proper goodness of fit tests is advisable. Furthermore, blocks will be required for large designs but these will not be discussed further here.

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*v 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 can construct a complete set of (v-1) orthogonal Latin squares of size v x v whenever v is a prime or a prime power while the GraecoLatin function can 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**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:

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

Prime p	2	3	5	7	11	13 ≤ 23	29 ≤ 97	97 ≤
Maximum q	13	8	6	5	4	3	2	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 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) x (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:

- any odd valued N
- any prime-power N = p**q where p and q can be chosen from Table 1
- any even valued N <= 30 except for 6 or 2

The GraecoLatin function generates even valued, non-prime power Graeco-Latin squares for N <= 30 by using the methods given by Street & Street (1987), Chapters 6 and 7.

The output is a single data frame of size (p**q) x (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 'blocksdesign' 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.

Lattice designs are constructed automatically by the 'blocksdesign' algorithm and will always attain the A-efficiency upper-bound for suitable sets of block design parameters. However, 'blocksdesign' does not distinguish or otherwise designate a design as a lattice design.

Tables 2 and 3 show the theoretical maximum number of replicates for square lattices with N <= 400 or rectangular lattices with N <= 380. Any design constructed by 'blocksdesign' which is a square or rectangular lattice will attain the theoretical A-efficiency upper bound provided that the number of replicates does not exceed the theoretical bound.

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 usually 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 is natural to consider modelling row and column effects 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. Example 11 shows an analysis assuming a Rows factor with 16 levels crossed with three nested columns factors Col1, Col2 and Col3 with 4, 8 and 34 levels, respectively.

****

Example 11

****

```
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

```
$Blocks_model

                First_order effects  D.Effic  A.Effic
                     (Reps)       1 1.000000 1.000000
                (Reps+Rows)      15 0.965741 0.953584
           (Reps+Rows+Col1)      18 0.961411 0.948560
      (Reps+Rows+Col1+Col2)      22 0.951867 0.936238
 (Reps+Rows+Col1+Col2+Col3)      48 0.888650 0.852710


                 Second_order effects  D.Effic  A.Effic
                     (Reps)^2      1       NA       NA
                (Reps+Rows)^2     15       NA       NA
           (Reps+Rows+Col1)^2     63 0.843056 0.778352
       Reps+Rows+Col1+Col2)^2    127 0.676124 0.538726
 (Reps+Rows+Col1+Col2+Col3)^2    543       NA       NA
```

****

A difficulty with the design in Example 11 is that, because the rows have 34 plots, not all of the nested column blocks can be of equal width within each level of nesting; two column blocks from each of Col1, Col2 and Col3 must be an extra plot wide. An alternative design could have been based on rows with 32 plots which would have allowed nested column blocks of equal width within each level of nesting. However, this would have meant that two replicates of 272 treatments would have required 17 rows which would have meant that the design could not be split into two replicates based on complete rows.

Example 12, shows how 'blocksdesign' can be used to construct a nested columns design with nested columns crossed with 17 rows of 32 plots where rows 1:8 and the first 16 plots of row 9 contain the first complete replicate and rows 10:17 and the last 16 plots of row 9 contain the second complete replicate. Note that the 'Reps' factor in Example 12 includes an additional factor level, Level 2, for row 9.

****

Example 12

****

```
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)
design(treatments, blocks, searches=1)
```

Output

```
$Blocks_model

                 First_order effects  D.Effic  A.Effic
                      (Reps)       2 0.997552 0.996636
                 (Reps+Rows)      16 0.963344 0.950485
            (Reps+Rows+Col1)      19 0.959030 0.945510
       (Reps+Rows+Col1+Col2)      23 0.949511 0.933271
  (Reps+Rows+Col1+Col2+Col3)      47 0.891292 0.856549


                 Second_order effects  D.Effic  A.Effic
                     (Reps)^2      2       NA       NA
                (Reps+Rows)^2     16       NA       NA
           (Reps+Rows+Col1)^2     67 0.832584 0.762761
      (Reps+Rows+Col1+Col2)^2    135 0.655904 0.513222
 (Reps+Rows+Col1+Col2+Col3)^2    543       NA       NA
```

****

Example 12 is likely to give a better design for control of row-and-column block effects than Example 11 and shows that the layout of row-and-column designs for multi-level nesting may require careful planning and some compromise for an effective design layout.

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.