Notes on the use of `dae` for design

C. J. Brien

March 26, 2023

Contents

1 Introduction

The R (R Core Team, 2023) package `dae` (Brien, 2023b) provides functions useful in the design and anova of experiments. This document describes how to use some of them to produce layouts for experiments and to check some of their properties. Brien et al. (2023) provide a general discussion of a paradigm for designing experiments that utilizes `dae` functions.

1.1 Functions to be used

The functions in `dae` fall into the following categories and those that will be covered in this document are listed and described:

1. Data

   BIBDWheat.dat Data for a balanced incomplete block experiment.

   Casuarina.dat Data for an experiment with rows and columns from Williams et al. (2002).

   Cabinet1.des A design for one of the growth cabinets in an experiment with 50 lines and 4 harvests.

   Exp249.mplot.des Systematic, main-plot design for an experiment to be run in a greenhouse.

   Fac4Proc.dat Data for a 2^4 factorial experiment.

   LatticeSquare_t49.des A Lattice square design for 49 treatments.

   McIntyreTMV.dat The design and data from McIntyre (1955) two-phase experiment.

   Oats.dat Data for an experiment to investigate nitrogen response of 3 oats varieties from Yates (1937).

   Sensory3Phase.dat Data for the three-phase sensory evaluation experiment in Brien and Payne (1999).

   Sensory3PhaseShort.dat Data for the three-phase sensory evaluation experiment in Brien and Payne (1999), but with short factor names.

   SPLGrass.dat Data for an experiment to investigate the effects of grazing patterns on pasture composition.

2. Factor manipulation functions

   fac.gen: Generate all combinations of several factors and, optionally, replicate them.

   fac.recast: Recasts a factor by modifying the values in the factor vector and/or the levels attribute, possibly combining some levels into a single level.

   fac.uselogical: Forms a two-level factor from a logical object.

   fac.combine: Combines several factors into one.

   fac.divide: Divides a factor into several separate factors.

fac.multinested: Creates several factors, one for each level of a nesting.fac and each of whose values are either generated within those of the level of nesting.fac or using the values of a nested.fac.

fac.nested: Creates a factor, the nested factor, whose values are generated within those of a nesting factor.

3. Design functions

designAnatomy: Given the layout for a design, obtain its anatomy via the canonical analysis of its projectors to show the confounding and aliasing inherent in the design.

designLatinSqrSys: Generate a systematic plan for a Latin Square design.

designBlocksGGPlot: Adds block boundaries to a plot produced by designGGPlot.

designGGPlot: A graphical representation of an experimental design based on labels stored in a data.frame using ggplot2.

designRandomize: Takes a systematic design and randomizes it according to the nesting (and crossing) relationships between the recipient(unit) factors for the randomization.

no.reps: Computes the number of replicates for an experiment.

summary.pcanon: Summarizes the anatomy of a design, being the decomposition of the sample space based on its canonical analysis, as produced by designAnatomy. The table produced includes the degrees of freedom and summary statistics of the canonical efficiency factors.

efficiencies.pcanon: Extracts the canonical efficiency factors from a pcanon.object produced by designAnatomy.

4. ANOVA functions

5. Matrix functions

6. Projector and canonical efficiency functions

efficiencies.pcanon: Produces a list containing the canonical efficiency factors for the joint decomposition of two or more sets of projectors (Brien and Bailey, 2009) obtained using `designAnatomy`.

7. Miscellaneous functions.

Documentation for these functions is available from the user manual via `vignette("dae-manual", package="dae")` and their are some notes that show how to use some of them in `vignette("DesignNotes", package="dae")`.

1.2 The paradigm

Fundamental to the approach in this document, and to using the functions described, is that a single allocation involves allocating a set of *allocated* factors to a set of *recipient* factors. In many designs, this allocation is achieved by randomization. However, sometimes there is systematic allocation or restricted allocation.

1.3 Notation used for mixed models

The general form for a mixed model is:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{u} + \mathbf{e},$$

where $\boldsymbol{\beta}$ is the vector of fixed parameters, $\mathbf{u}$ is the vector of random effects, and $\mathbf{e}$ is the vector of residuals corresponding to each observation. The matrices $\mathbf{X}$ and $\mathbf{Z}$ are the design matrices for the fixed and random effects, respectively. Generally, $\mathbf{X}$ and $\boldsymbol{\beta}$ are conformably partitioned so that there is a separate submatrix and subvector for each fixed term. Similarly, $\mathbf{Z}$ and $\mathbf{u}$ are conformably partitioned according to the random terms.

A mixed model is expressed in symbolic form by list of the fixed terms, followed by a '|', and then a list of the random terms. Terms contributing to the residuals are underlined.

2 Single-allocation orthogonal design in R

This class of experiments covers the orthogonal standard or textbook experiments and these experiments must be single phase because they involve a single randomization, in the sense that the randomization can be achieved with a single permutation. Hence there will be two sets of factors, or tiers, one set being allocated to the other set. In `designRandomize`, these are referred to as the allocated and recipient sets of factors. They are also called the unit and treatment factors, respectively.

Firstly, initialize by loading the `dae` library. Also check the version that is loaded.

```
library(dae)

## Loading required package:  ggplot2

packageVersion("dae")

## [1] '3.2.15'
```

2.1 Two potential designs for a 5×5 grid of plots

Suppose an experiment to investigate five treatments is to be conducted on 25 plots, the 25 plots being arranged in a 5×5 grid. Two possible designs are a randomized complete-block design (RCBD) or a Latin square design (LSqD). The factor-allocation diagram (Brien et al., 2011) for the RCBD is in Figure 1 and that for the LSqD is in Figure 2.

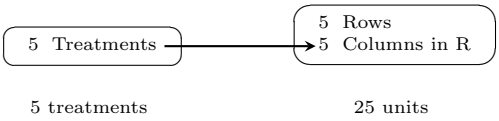


Figure 1: Factor-allocation diagram for an RCBD: treatments are allocated to units; the arrow indicates that the factor Treatments is randomized to Columns; Columns in R indicates that the Columns are considered to be nested within Rows for this randomization; R = Rows.

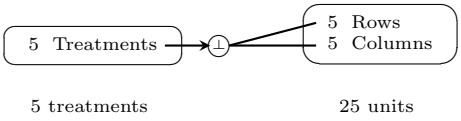


Figure 2: Factor-allocation diagram for an LSqD: treatments are allocated to units; the arrow indicates that the allocation is randomized; the '⊥' at the end of the arrow indicates that an orthogonal design is used; the two lines from '⊥' indicates that the Treatments are allocated to the combinations of Rows and Columns using the design.

2.1.1 Produce the randomized layout for an RCBD

Use `designRandomize` to randomize the treatments according to an RCBD. The arguments to `designRandomize` that need to be set are (i) `allocated`, (ii) `nested.recipients`, (iii) `recipient`, and optionally, (iv) `seed`. The allocated factors are also referred to as treatment factors and the recipient factors as block or unit factors. A systematic arrangement of the allocated factors, corresponding to the values of the recipient factors, needs to be supplied and there are a number of ways of doing this.

In these notes, the general approach is to set up a systematic design in a `data.frame` to separate this aspect of constructing a design from the randomizing of a design. The naming convention used is that the name of the `data.frame` containing the systematic design ends ends in `.sys`. This `data.frame` should contain the values of both the recipient and the allocated `factors`, the latter in a systematic order that is appropriate for the design. The `dae` function `fac.gen` will be used to generate the values of the recipient `factors` in standard order and often will also be used to generate the values of the allocated `factors`.

Then the allocated and recipient factors are supplied to `designRandomize` by subsetting the columns of the `data.frame`s to just the appropriate factors for each argument. Note that the Treatments could also be supplied as a factor and the recipient factors can be specified directly to the `recipient` argument as a `list`, e.g. `list(Rows=b, Columns=t)`. A `data.frame` containing the recipient and randomized allocated factors is produced and the name for the `data.frame` with the randomized layout will end in `.lay`.

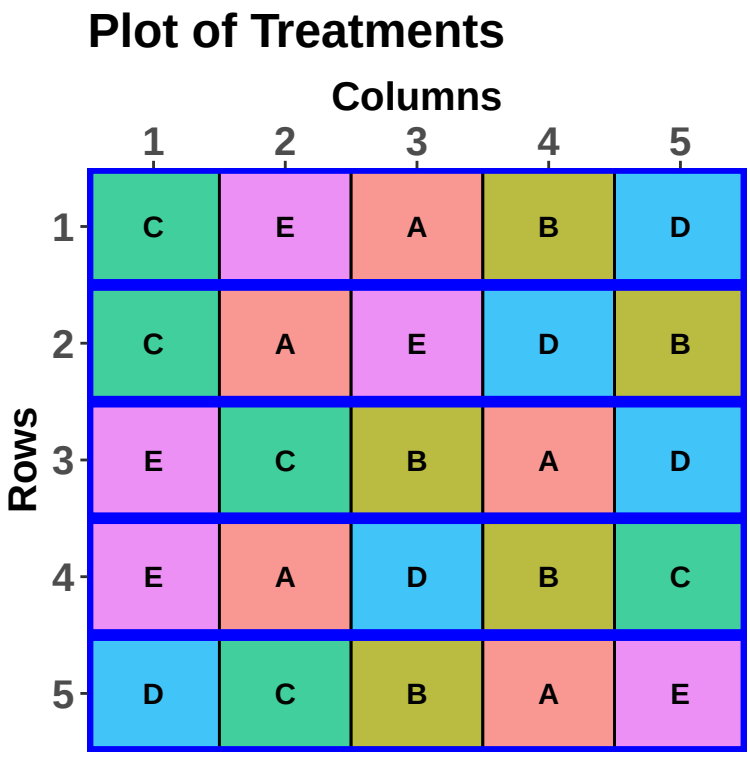
The randomization is controlled by `nested.recipients`: nested `recipient` factors are permuted within those factors that nest them. Only the nesting is specified: it is assumed that if two factors are not nested then they must be crossed. So for this example, given that the `nested.recipients` has Columns nested within Rows, the randomized layout is obtained by permuting (i) Rows and (ii) Columns within Rows. Then the permuted Rows and Columns and the systematic Treatments are sorted so that Rows and Columns are in standard order.

In this example, the allocated factor is Treatments, with 5 levels, and the recipient factors are Rows and Columns, both with 5 levels. Suppose that Rows are to form the blocks.

Use the following R code to obtain and display the layout:

```
#'## Obtain the layout
b <- 5
t <- 5
#'## Set up a systematic design
RCBD.sys <- cbind(fac.gen(generate = list(Rows=b, Columns=t)),
                  fac.gen(generate = list(Treatments = LETTERS[1:t]),
                          times = b))
#'## Obtain the layout
RCBD.lay <- designRandomize(allocated         = RCBD.sys["Treatments"],
                            recipient         = RCBD.sys[c("Rows", "Columns")],
                            nested.recipients = list(Columns = "Rows"),
                            seed              = 1134)

#'## Output the layout
RCBD.lay

##    Rows Columns Treatments
## 1     1       1          C
## 2     1       2          E
## 3     1       3          A
## 4     1       4          B
## 5     1       5          D
## 6     2       1          C
## 7     2       2          A
## 8     2       3          E
## 9     2       4          D
## 10    2       5          B
## 11    3       1          E
## 12    3       2          C
## 13    3       3          B
## 14    3       4          A
## 15    3       5          D
## 16    4       1          E
## 17    4       2          A
## 18    4       3          D
## 19    4       4          B
## 20    4       5          C
## 21    5       1          D
## 22    5       2          C
## 23    5       3          B
```

```
## 24    5       4          A
## 25    5       5          E

#'## Plot the layout
designGGPlot(RCBD.lay, labels = "Treatments", cellalpha = 0.75,
             blockdefinition = cbind(1,t))
```



The function `fac.gen` is from the package `dae` and generates the factors in the `list` in standard order with the specified numbers of levels or the levels in supplied character or numeric vectors. The `seed` is specified to ensure that the same design is produced whenever `designRandomize` is run with these arguments.

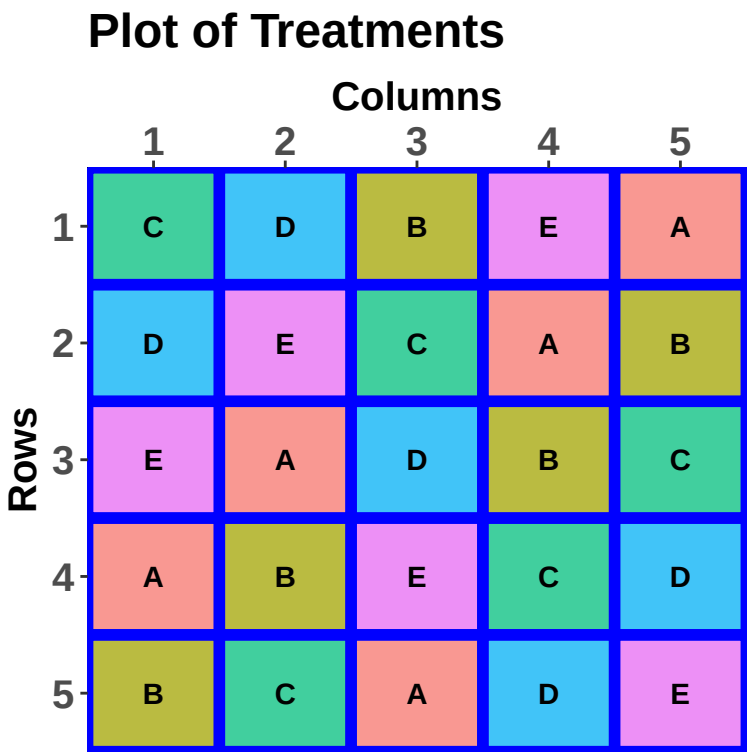
2.1.2 Produce the randomized layout for an LSqD

Use `designRandomize` to randomize the treatments according to an LSqD, having obtained the systematic design using `fac.gen` and `designLatinSqrSys`. For this design, Rows and Columns are crossed; there are no nested factors. Consequently, the `nested.recipients` argument is omitted and `designRandomize` assumes that the `recipient` factors are crossed. The layout can be obtained using the following R code:

```
b <- 5
t <- 5
#'## Set up a systematic design
LSqD.sys <- cbind(fac.gen(list(Rows=b, Columns=t)),
                  Treatments = factor(designLatinSqrSys(t), labels = LETTERS[1:t]))
#'## Obtain the layout
LSqD.lay <- designRandomize(allocated = LSqD.sys["Treatments"],
                            recipient = LSqD.sys[c("Rows", "Columns")],
                            seed      = 141)
#'## Output the layout
LSqD.lay

##    Rows Columns Treatments
```

```
## 1     1       1          C
## 2     1       2          D
## 3     1       3          B
## 4     1       4          E
## 5     1       5          A
## 6     2       1          D
## 7     2       2          E
## 8     2       3          C
## 9     2       4          A
## 10    2       5          B
## 11    3       1          E
## 12    3       2          A
## 13    3       3          D
## 14    3       4          B
## 15    3       5          C
## 16    4       1          A
## 17    4       2          B
## 18    4       3          E
## 19    4       4          C
## 20    4       5          D
## 21    5       1          B
## 22    5       2          C
## 23    5       3          A
## 24    5       4          D
## 25    5       5          E
```

```
#'## Plot the layout
designGGPlot(LSqD.lay, labels = "Treatments", cellalpha = 0.75,
             blockdefinition = cbind(1,1))
```



2.1.3 Check the properties of the designs

The properties of the designs can be investigated using `designAnatomy`.

Because these experiments involve a single randomization, they are two-tiered. That is, there are just two sets of factors involved in the randomization. As we have seen, the first set of factors is the set of allocated (treatment) factors and the second set is the set of recipient (unit) factors. Further there will be a set of projectors associated with each tier and `designAnatomy` is used to do an eigenanalysis of the relationships between the two sets of projectors. The sets of projectors are specified to `designAnatomy` via model `formulae`, the formula for the recipient factors coming first in the `list` for `formulae`.

For both the RCBD and LSqD the two sets of factors are (i) {Rows, Columns} and (ii) {Treatments}. What differs between the two designs is the nesting/crossing relationship between Rows and Columns and this will be expressed in the `formulae`.

Use the commands given below to produce the anatomies (skeleton anova tables) for the RCBD and LSqD that have been obtained. Note that the 'Mean' source has been omitted from these tables, but can be included using `grandMean = TRUE` when calling `designAnatomy`.

```
#'## Anatomy for the RCBD
RCBD.canon <- designAnatomy(formulae = list(unit = ~ Rows/Columns,
                                            trt  = ~ Treatments),
                            data     = RCBD.lay)
summary(RCBD.canon)

##
##
## Summary table of the decomposition for unit & trt
##
##  Source.unit   df1 Source.trt df2 aefficiency eefficiency order
##  Rows            4
##  Columns[Rows]  20 Treatments   4      1.0000      1.0000     1
##                    Residual    16

#'## Anatomy for the LSqD
LSqD.canon <- designAnatomy(formulae = list(unit = ~ Rows*Columns,
                                            trt  = ~ Treatments),
                            data     = LSqD.lay)
summary(LSqD.canon)

##
##
## Summary table of the decomposition for unit & trt
##
##  Source.unit  df1 Source.trt df2 aefficiency eefficiency order
##  Rows           4
##  Columns        4
##  Rows#Columns  16 Treatments   4      1.0000      1.0000     1
##                   Residual    12
```

Get the mixed-model terms for the analysis by rerunning the summary function with the `labels.swap` argument set to `TRUE`.

```
#'## Term-based anatomy for the RCBD
summary(RCBD.canon, labels.swap = TRUE)

##
##
```

```
## Summary table of the decomposition for unit & trt
##
##  Term.unit    df1 Term.trt   df2 aefficiency eefficiency order
##  Rows           4
##  Rows:Columns  20 Treatments   4      1.0000      1.0000     1
##                   Residual    16

#'## Term-based anatomy for the LSqD
summary(LSqD.canon, labels.swap = TRUE)

##
##
## Summary table of the decomposition for unit & trt
##
##  Term.unit    df1 Term.trt   df2 aefficiency eefficiency order
##  Rows           4
##  Columns        4
##  Rows:Columns  16 Treatments   4      1.0000      1.0000     1
##                   Residual    12
```

2.1.4 Questions

1. What is the advantage of specifying a `seed` in `designRandomize`?

   It means that the design can be reproduced in subsequent executions of the R script.

2. With what unit source is Treatments confounded in these designs and what is the difference in the interpretation of these sources?

   Treatments is confounded with the term Rows:Columns. For the RCBD, Treatments is confounded with the source Columns[Rows]. For the LSqD, Treatments is confounded with the source Rows#Columns. The source Columns[Rows] reflects the differences between Rows within Columns; Rows#Columns is the interaction of Rows-and-Columns and reflects how the differences between Rows (Columns) vary between Columns (Rows).

3. What would determine which of these two designs is used for a particular experiment?

   In a discussion with the researcher, it needs to be determined whether overall Column differences can be ruled out. If they can, then the RCBD should be used; otherwise, the LSqD would be used.

2.2 Split-plot from Yates (1937)

Yates (1937) describes a split-plot experiment that investigates the effects of three varieties of oats and four levels of Nitrogen fertilizer. The varieties are assigned to the main plots using a randomized complete-block design with 6 blocks and the nitrogen levels are randomly assigned to the subplots in each main plot. The factor-allocation diagram for the experiment is in Figure 3.

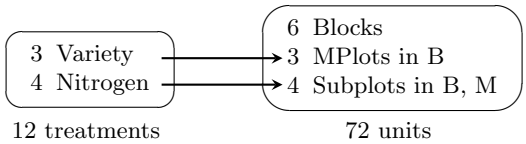


Figure 3: Factor-allocation diagram for a split-plot design: treatments are allocated to units; the arrows indicates that the factors Variety and Nitrogen are randomized to MPlots and Subplots, respectively; MPlots in B indicates that the MPlots are considered to be nested within Blocks for this randomization; Subplots in B, M indicates that the Subplots are considered to be nested within Blocks and MPlots for this randomization; B = Blocks, M = MPlots

2.2.1 Produce the randomized experimental layout

Use `fac.gen` to obtain a systematic layout and then `designRandomize` to obtain a randomized layout for this experiment. Check the properties of the design, as illustrated in the following R code:

```
Oats.sys <- cbind(fac.gen(list(Blocks=6, MPlots=3, SubPlots=4)),
                  fac.gen(list(Variety=c("Victory","Golden Rain", "Marvellous"),
                               Nitrogen=c(0,0.2,0.4,0.6)), times=6))
Oats.lay <- designRandomize(allocated         = Oats.sys[c("Variety", "Nitrogen")],
                            recipient         = Oats.sys[c("Blocks", "MPlots", "SubPlots")],
                            nested.recipients = list(MPlots = "Blocks",
                                                     SubPlots = c("MPlots", "Blocks")),
                            seed              = 235805)
#'## Display design produced
Oats.lay
```

```
##    Blocks MPlots SubPlots     Variety Nitrogen
## 1       1      1        1  Marvellous      0.4
## 2       1      1        2  Marvellous        0
## 3       1      1        3  Marvellous      0.2
## 4       1      1        4  Marvellous      0.6
## 5       1      2        1     Victory        0
## 6       1      2        2     Victory      0.2
## 7       1      2        3     Victory      0.6
## 8       1      2        4     Victory      0.4
## 9       1      3        1 Golden Rain      0.2
## 10      1      3        2 Golden Rain      0.4
## 11      1      3        3 Golden Rain      0.6
## 12      1      3        4 Golden Rain        0
## 13      2      1        1  Marvellous      0.4
## 14      2      1        2  Marvellous      0.2
## 15      2      1        3  Marvellous        0
## 16      2      1        4  Marvellous      0.6
## 17      2      2        1     Victory      0.2
## 18      2      2        2     Victory        0
## 19      2      2        3     Victory      0.6
## 20      2      2        4     Victory      0.4
## 21      2      3        1 Golden Rain      0.6
## 22      2      3        2 Golden Rain      0.4
## 23      2      3        3 Golden Rain      0.2
## 24      2      3        4 Golden Rain        0
## 25      3      1        1 Golden Rain      0.2
## 26      3      1        2 Golden Rain      0.6
## 27      3      1        3 Golden Rain      0.4
## 28      3      1        4 Golden Rain        0
## 29      3      2        1  Marvellous      0.4
## 30      3      2        2  Marvellous      0.6
## 31      3      2        3  Marvellous        0
## 32      3      2        4  Marvellous      0.2
## 33      3      3        1     Victory      0.4
## 34      3      3        2     Victory      0.2
## 35      3      3        3     Victory        0
## 36      3      3        4     Victory      0.6
## 37      4      1        1  Marvellous        0
## 38      4      1        2  Marvellous      0.4
```

```
## 39      4      1        3  Marvellous      0.2
## 40      4      1        4  Marvellous      0.6
## 41      4      2        1 Golden Rain      0.2
## 42      4      2        2 Golden Rain      0.6
## 43      4      2        3 Golden Rain      0.4
## 44      4      2        4 Golden Rain        0
## 45      4      3        1     Victory      0.4
## 46      4      3        2     Victory        0
## 47      4      3        3     Victory      0.6
## 48      4      3        4     Victory      0.2
## 49      5      1        1 Golden Rain      0.2
## 50      5      1        2 Golden Rain        0
## 51      5      1        3 Golden Rain      0.6
## 52      5      1        4 Golden Rain      0.4
## 53      5      2        1  Marvellous        0
## 54      5      2        2  Marvellous      0.2
## 55      5      2        3  Marvellous      0.6
## 56      5      2        4  Marvellous      0.4
## 57      5      3        1     Victory      0.4
## 58      5      3        2     Victory      0.2
## 59      5      3        3     Victory      0.6
## 60      5      3        4     Victory        0
## 61      6      1        1  Marvellous        0
## 62      6      1        2  Marvellous      0.6
## 63      6      1        3  Marvellous      0.4
## 64      6      1        4  Marvellous      0.2
## 65      6      2        1     Victory      0.4
## 66      6      2        2     Victory      0.2
## 67      6      2        3     Victory        0
## 68      6      2        4     Victory      0.6
## 69      6      3        1 Golden Rain      0.6
## 70      6      3        2 Golden Rain      0.2
## 71      6      3        3 Golden Rain      0.4
## 72      6      3        4 Golden Rain        0
```

```
#'## Check its properties
Oats.canon <- designAnatomy(formulae = list(unit = ~ Blocks/MPlots/SubPlots,
                                            trt  = ~ Variety*Nitrogen),
                            data     = Oats.lay)
summary(Oats.canon, which.criteria = c("aeff", "order"))
```

```
##
##
## Summary table of the decomposition for unit & trt
##
##  Source.unit             df1 Source.trt       df2 aefficiency order
##  Blocks                    5
##  MPlots[Blocks]           12 Variety            2      1.0000     1
##                              Residual          10
##  SubPlots[Blocks:MPlots]  54 Nitrogen           3      1.0000     1
##                              Variety#Nitrogen   6      1.0000     1
##                              Residual          45
```

2.2.2 Analysis of variance (anova) for the Yields

After reading in the data, use the `aov` function to produce the anova as shown below. Note the use of the `Error` function to produce two Residual lines, one each for Wplots and Subplots (Note the change from MPlots to Wplots).

```
#'## Read in data for actual experiment
data("Oats.dat")

#'## Analyse by anova
oats.aov <- aov(Yield ~ Nitrogen*Variety +
                  Error(Blocks/Wplots/Subplots), data=Oats.dat)
summary(oats.aov)

##
## Error: Blocks
##           Df Sum Sq Mean Sq F value Pr(>F)
## Residuals  5  15875    3175
##
## Error: Blocks:Wplots
##           Df Sum Sq Mean Sq F value Pr(>F)
## Variety    2   1786   893.2   1.485  0.272
## Residuals 10   6013   601.3
##
## Error: Blocks:Wplots:Subplots
##                  Df Sum Sq Mean Sq F value   Pr(>F)
## Nitrogen          3  20021    6674  37.686 2.46e-12 ***
## Nitrogen:Variety  6    322      54   0.303    0.932
## Residuals        45   7969     177
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The anova table shown here is the same as the anatomy, but in a different format.

2.2.3 Questions

1. In what sense does this design involve a single randomization?

   In the sense that the randomization of both Nitrogen and Variety can be achieved with a single permutation of the units, the subplots.

2. What is the initial allocated mixed model for this design? Is it equivalent to a randomization model?

   The initial allocation mixed model is Variety + Nitrogen + Variety^Nitrogen | Blocks + Blocks^MPlots + Blocks^MPlots^SubPlots. The initial allocation model is equivalent to a randomization model because the allocation was a randomization.

3. A factorial RCBD would involve randomizing the $3 \times 4 = 12$ treatments to the 12 subplots within each block. What has been achieved in using the split-plot design as compared to a factorial RCBD?

   The precision of the Variety differences has been sacrificed to increase the precision of the Nitrogen differences. This is the case because the Residual mean square for MPlots[Blocks] is substantially larger than that for Subplots[Blocks^MPlots]. If a factorial RCBD had been used, the Residual mean square for Plots[Blocks] would be the weighted average of the two Residual mean squares from the split-plot experiment, the weight being the Residual degrees of freedom. That is, the value of the Residual mean square for the factorial RCBD would be between the values for the two Residual mean squares for the split-plot design.

2.3 A design for a petrol additives experiment

Box et al. (2005, Section 4.4) describes a car emission experiment that investigates 4 additives. It involves 4 cars being driven by 4 drivers. Here we investigate increasing the replication by repeating the experiment on two occasions. Suppose that the 4 cars differ between occasions.

In a `data.frame` called `LSRepeat.sys`, generate a systematic design using two 4×4 Latin squares to for allocating the 4 Additives to the 32 tests, being the combinations of the 2 Occasions x 4 Drivers x 4 Cars.

Now a comparison is made of two different ways of randomizing this design. Firstly, we retain the factors Occasions, Drivers and Cars from the systematic design. The factor-allocation diagram is in Figure 4.

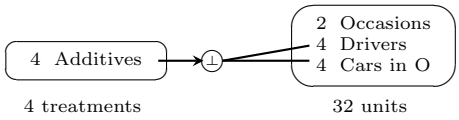
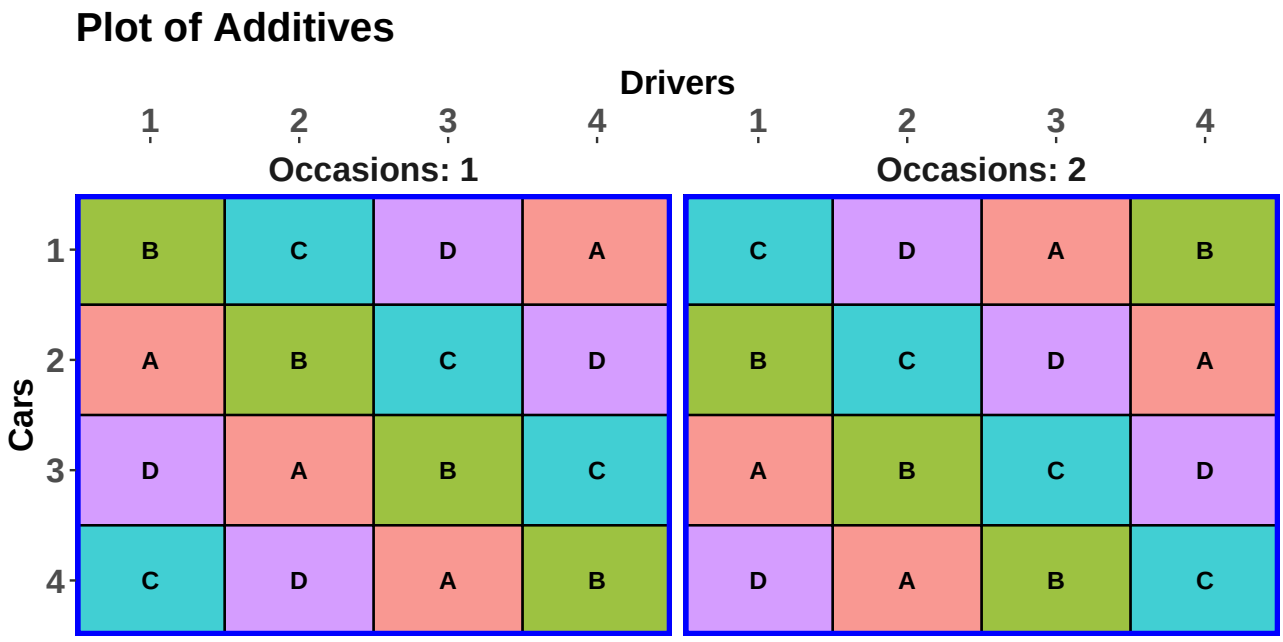


Figure 4: Factor-allocation diagram for repeated LSqDs: treatments are allocated to units; the arrow indicates that the allocation is randomized; the '⊥' at the end of the arrow indicates that an orthogonal design is used; the two lines from '⊥' indicates that the Additives are allocated to the combinations of Drivers and Cars within Occasions using the design.

```
#'## Obtain a randomized layout with Cars nested within Occasions
LSRepeat2b.lay <- designRandomize(allocated         = LSRepeat.sys["Additives"],
                                  recipient         = LSRepeat.sys[c("Occasions", "Drivers",
                                                                     "Cars")],
                                  nested.recipients = list(Cars="Occasions"),
                                  seed              = 194)
LSRepeat2b.lay

##    Occasions Drivers Cars Additives
## 1          1       1    1         B
## 2          1       1    2         A
## 3          1       1    3         D
## 4          1       1    4         C
## 5          1       2    1         C
## 6          1       2    2         B
## 7          1       2    3         A
## 8          1       2    4         D
## 9          1       3    1         D
## 10         1       3    2         C
## 11         1       3    3         B
## 12         1       3    4         A
## 13         1       4    1         A
## 14         1       4    2         D
## 15         1       4    3         C
## 16         1       4    4         B
## 17         2       1    1         C
## 18         2       1    2         B
## 19         2       1    3         A
## 20         2       1    4         D
## 21         2       2    1         D
## 22         2       2    2         C
## 23         2       2    3         B
## 24         2       2    4         A
## 25         2       3    1         A
## 26         2       3    2         D
## 27         2       3    3         C
```

```
## 28          2       3    4          B
## 29          2       4    1          B
## 30          2       4    2          A
## 31          2       4    3          D
## 32          2       4    4          C

LSRepeat2b.canon <- designAnatomy(formulae = list(unit = ~ (Occasions/Cars)*Drivers,
                                                  trt = ~ Additives),
                                  data     = LSRepeat2b.lay)
summary(LSRepeat2b.canon)

##
##
## Summary table of the decomposition for unit & trt
##
##  Source.unit             df1 Source.trt df2 aefficiency eefficiency order
##  Occasions                 1
##  Cars[Occasions]           6
##  Drivers                   3
##  Occasions#Drivers         3
##  Cars#Drivers[Occasions]  18 Additives    3      1.0000      1.0000     1
##                              Residual    15

#'## Plot the layout
designGGPlot(LSRepeat2b.lay, row.factors = "Cars", column.factors = c("Occasions", "Drivers"),
             labels = "Additives", cellalpha = 0.75, blockdefinition = cbind(4,4))
```



Now we use only Drivers and Cars to do the randomization, but still attempt to include Occasions in the analysis. The new factor-allocation diagram is in Figure 5.

```
#'## Obtain a randomized layout
LSRepeat.D8.sys <- LSRepeat.sys
LSRepeat.D8.sys$Cars <- with(LSRepeat.D8.sys, fac.combine(list(Occasions, Cars)))
```

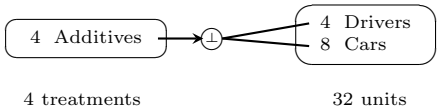
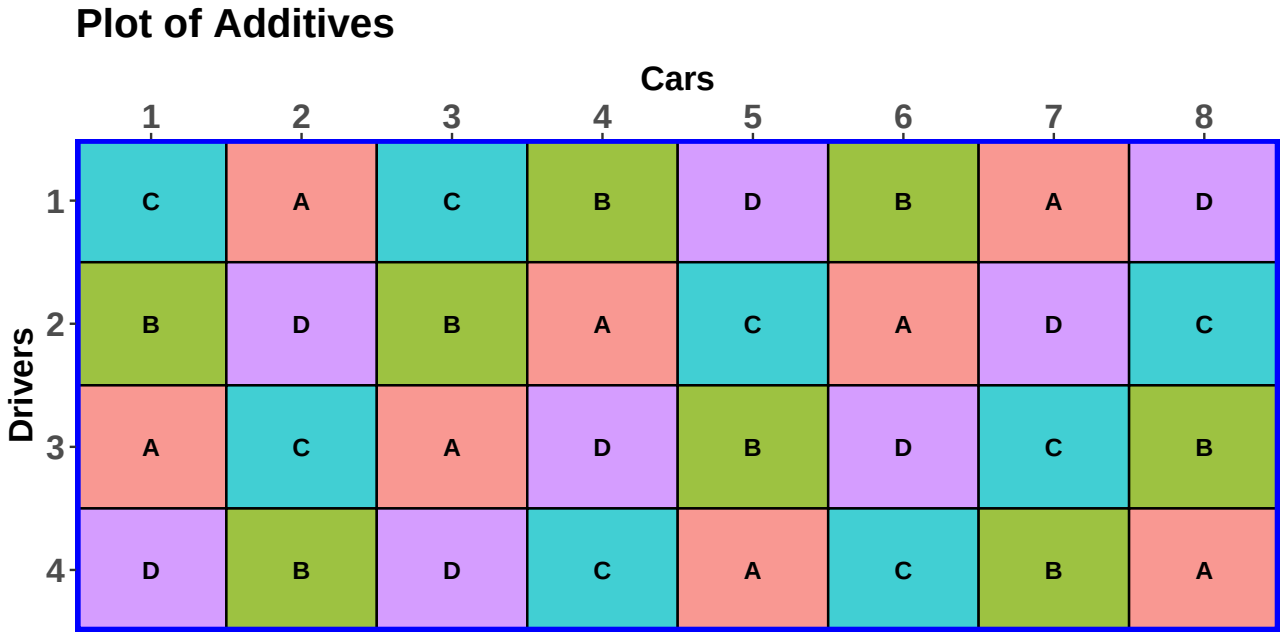


Figure 5: Factor-allocation diagram for repeated LSqDs: treatments are allocated to units; the arrow indicates that the allocation is randomized; the '⊥' at the end of the arrow indicates that an orthogonal design is used; the two lines from '⊥' indicates that the Additives are allocated to the combinations of Drivers and Cars using the design.

```
LSRepeat.D8.sys <- with(LSRepeat.D8.sys, LSRepeat.D8.sys[order(Drivers,Cars),])
LSRepeat2b.D8.lay <- designRandomize(allocated = LSRepeat.D8.sys["Additives"],
                                     recipient = LSRepeat.D8.sys[c("Drivers", "Cars")],
                                     seed      = 149)

#'## Plot the layout
designGGPlot(LSRepeat2b.D8.lay, row.factors = "Drivers", column.factors = "Cars",
             labels = "Additives", cellfillcolour.column = "Additives",
             cellalpha = 0.75, blockdefinition = cbind(4,8))
```




```
#'## Get the Anatomy of the layout
LSRepeat2.D8.canon <- designAnatomy(formulae = list(unit = ~ Drivers*Cars,
                                                    trt  = ~ Additives),
                                    data     = LSRepeat2b.D8.lay)
summary(LSRepeat2.D8.canon)

##
##
## Summary table of the decomposition for unit & trt
##
##  Source.unit  df1 Source.trt df2 aefficiency eefficiency order
##  Drivers        3
##  Cars           7
##  Drivers#Cars  21 Additives    3      1.0000      1.0000     1
```

```
##                   Residual    18
```

```
#'## Add Occasions to the analysis
LSRepeat2b.D8.lay$Occasions <- fac.recast(LSRepeat2b.D8.lay$Cars,
                                          newlevels = rep(1:2, each=4))
LSRepeat2b.D8.lay
```

```
##    Drivers Cars Additives Occasions
## 1        1    1         C         1
## 2        1    2         A         1
## 3        1    3         C         1
## 4        1    4         B         1
## 5        1    5         D         2
## 6        1    6         B         2
## 7        1    7         A         2
## 8        1    8         D         2
## 9        2    1         B         1
## 10       2    2         D         1
## 11       2    3         B         1
## 12       2    4         A         1
## 13       2    5         C         2
## 14       2    6         A         2
## 15       2    7         D         2
## 16       2    8         C         2
## 17       3    1         A         1
## 18       3    2         C         1
## 19       3    3         A         1
## 20       3    4         D         1
## 21       3    5         B         2
## 22       3    6         D         2
## 23       3    7         C         2
## 24       3    8         B         2
## 25       4    1         D         1
## 26       4    2         B         1
## 27       4    3         D         1
## 28       4    4         C         1
## 29       4    5         A         2
## 30       4    6         C         2
## 31       4    7         B         2
## 32       4    8         A         2
```

```
LSRepeat2b.D8.canon <- designAnatomy(formulae = list(unit = ~ (Occasions + Cars)*Drivers,
                                                     trt  = ~ Additives),
                                     data      = LSRepeat2b.D8.lay)
summary(LSRepeat2b.D8.canon)
```

```
##
##
## Summary table of the decomposition for unit & trt (based on adjusted quantities)
##
##  Source.unit             df1 Source.trt df2 aefficiency eefficiency order
##  Occasions                 1
##  Cars[Occasions]           6
##  Drivers                   3
##  Occasions#Drivers         3 Additives    3      0.1500      0.1250     2
```

```
##  Cars#Drivers[Occasions]  18 Additives    3      0.8289      0.7500     2
##                              Residual    15
##
## The design is not orthogonal
```

2.3.1 Questions

1. What is the difference between the two randomizations?

   For the first randomization, the Additives are randomized to the Cars within Occasions so that each Driver does all 4 Additives in the 4 Cars in an Occasion. The design is said to be resolved. This does not happen with the randomization based on only Drivers and Cars.

2. How do the two anatomies that include Occasions differ?

   The first anatomy is orthogonal and does not have any information about Additives confounded with Cars#Drivers[Occasions]. On the other hand, the second anatomy, based on the layout where Occasions was not included in the randomization, is not orthogonal. Additives information is partially confounded with both Occasions#Drivers and Cars#Drivers[Occasions].

3. What effect does including Occasions#Drivers have on the anatomy?

   Including Occasions#Drivers reduces the Residual DF by 3 (from 18 to 15).

3 Single-allocation, nonorthogonal design in R

This class of experiments covers the nonorthogonal standard or textbook experiments.

3.1 Twenty treatments in an alpha design

The following table gives an alpha design for 20 treatments, taken from Williams et al. (2002, p.128). The design has 3 replicates, each of which contains 5 blocks of 4 plots. It is a resolved design in that each replicate contains a complete set of the treatments.

Table 1: Unrandomized alpha design for 20 treatments

	Replicate														
	1					2					3				
Block	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
2	6	7	8	9	10	7	8	9	10	6	8	9	10	6	7
3	11	12	13	14	15	13	14	15	11	12	15	11	12	13	14
4	16	17	18	19	20	19	20	16	17	18	17	18	19	20	16

The factor-allocation diagram for the experiment is in Figure 6.

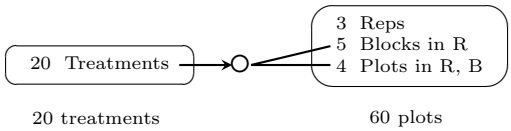


Figure 6: Factor-allocation diagram for the alpha design: treatments are allocated to units; the arrow indicates that the allocation is randomized; the 'O' at the end of the arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicate that the Treatments are allocated to the combinations of Blocks and Plots using the design; Blocks in R indicates that the Blocks are considered to be nested within Reps for this randomization; Plots in R, B indicates that the Plots are considered to be nested within Reps and Blocks for this randomization; R = Reps; B = Blocks.

3.1.1 Produce the randomized layout for the alpha design and check its properties

Use `designRandomize` to obtain the randomized layout and `designAnatomy` to check its properties.

```
#'## Set up the systematic design
# Note that Treatments has been entered by rows within a replicate
alpha.sys <- cbind(fac.gen(list(Reps=3, Plots=4, Blocks=5)),
                   Treats = factor(c(1:20,
                                     1:5, 7:10,6, 13:15,11,12, 19,20,16:18,
                                     1:5, 8:10,6,7, 15,11:14, 17:20,16)))
#'## Obtain the layout
alpha.lay <- designRandomize(allocated         = alpha.sys["Treats"],
                             recipient         = alpha.sys[c("Reps", "Plots", "Blocks")],
                             nested.recipients = list(Blocks = "Reps",
                                                      Plots = c("Reps", "Blocks")),
                             seed              = 918508)
alpha.lay <- with(alpha.lay, alpha.lay[order(Reps,Blocks,Plots), ])

#'## Check its properties
alpha.canon <- designAnatomy(formulae       = list(units = ~ Reps/Blocks/Plots,
```

```
                                                  trts  = ~ Treats),
                           which.criteria = "all",
                           data           = alpha.lay)
summary(alpha.canon, which.criteria = "all")

##
##
## Summary table of the decomposition for units & trts (based on adjusted quantities)
##
##  Source.units       df1 Source.trts df2 aefficiency eefficiency mefficiency sefficiency xefficiency
##  Reps                 2
##  Blocks[Reps]        12 Treats       12      0.2778      0.1667      0.3333      0.0152      0.4167
##  Plots[Reps:Blocks]  45 Treats       19      0.7447      0.5833      0.7895      0.0365      1.0000
##                         Residual     26
##  order dforthog
##
##      2        0
##      3        7
##
##
## The design is not orthogonal
```

The summary table shows us a number of summary statistics calculated from the canonical efficiency factors. They are:

aefficiency: the harmonic mean of the nonzero canonical efficiency factors.

mefficiency: the mean of the nonzero canonical efficiency factors.

eefficiency: the minimum of the nonzero canonical efficiency factors.

sefficiency: the variance of the nonzero canonical efficiency factors.

xefficiency: the maximum of the nonzero canonical efficiency factors.

order: the order of balance and is the number of unique nonzero canonical efficiency factors.

dforthog: the number of canonical efficiency factors that are equal to one.

For this example it can be seen that (i) an average 74.47%, as measured by the harmonic mean, or 78.95%, as measured by the arithmetic mean, of the information about Treats is confounded with the differences between plots within the reps-blocks combinations and (ii) there are 3 different efficiency factors associated with the 19 Treats degrees of freedom estimated from Plots[Reps:Blocks], the smallest of which is 0.5833 and 7 of which are one. In this case, where the treatments are equally replicated, it can be concluded that the mean variance of a normalized treatment contrast is inversely proportional to the harmonic mean of the canonical efficiency factors, that is, to 0.7447.

Get the mixed-model terms for the analysis by rerunning the summary function with the `labels.swap` argument set to `TRUE`.

```
#'## Obtain the terms for the design
summary(alpha.canon, which.criteria = "all", labels.swap = TRUE)

##
##
## Summary table of the decomposition for units & trts (based on adjusted quantities)
##
##  Term.units        df1 Term.trts df2 aefficiency eefficiency mefficiency sefficiency xefficiency
```

```
##  Reps                2
##  Reps:Blocks        12 Treats     12      0.2778      0.1667      0.3333      0.0152      0.4167
##  Reps:Blocks:Plots  45 Treats     19      0.7447      0.5833      0.7895      0.0365      1.0000
##                        Residual   26
##  order dforthog
##
##      2        0
##      3        7
##
##
## The design is not orthogonal
```

3.1.2 Questions

1. What is the randomization-based mixed model for this experiment?

   The trts term (Source.trts) provides the fixed term and the units terms (Source.units) provide the random terms. Hence, the symbolic, randomization-based, mixed model is Treats | Reps + Reps^Blocks + Reps^Blocks^Plots.

2. In a mixed-model analysis, which unit terms might you fit as fixed terms? Why?

   Reps is a definite candidate for the following reasons. Firstly, Reps has only two degrees of freedom and it will be difficult to estimate a variance component for it. Secondly, one does not want to estimate Treats from Reps (there is no Treats information between Reps).

3.2 Balanced incomplete-block design from Joshi (1987)

Joshi (1987) gives an experiment to investigate six varieties of wheat that employs a balanced incomplete-block design with 10 blocks, each consisting of three plots. The factor-allocation diagram for the experiment is in Figure 7.

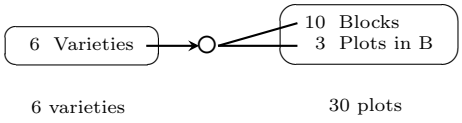


Figure 7: Factor-allocation diagram for the balanced incomplete-block design: treatments are allocated to units; the arrow indicates that the allocation is randomized; the 'O' at the end of the arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicates that the Varieties are allocated to the combinations of Blocks and Plots using the design; Plots in B indicates that the Plots are considered to be nested within Blocks for this randomization; B = Blocks.

3.2.1 Input the Yields and check properties of the design

Use the following R code to input the data for the experiment and check its properties.

```
#'## Input the design and data
data("BIBDWheat.dat")

#'## Check the properties of the design
bibdwheat.canon <- designAnatomy(formulae = list(units = ~ Blocks/Plots,
                                                 trts  = ~ Varieties),
                                 data     = BIBDWheat.dat)
summary(bibdwheat.canon)
```

```
##
##
## Summary table of the decomposition for units & trts (based on adjusted quantities)
##
##  Source.units  df1 Source.trts df2 aefficiency eefficiency order
##  Blocks          9 Varieties     5      0.2000      0.2000     1
##                    Residual      4
##  Plots[Blocks]  20 Varieties     5      0.8000      0.8000     1
##                    Residual     15
##
## The design is not orthogonal
```

From this it is clear that 80% of the information about Varieties is available from the Plots[Blocks] source; that is, 80% of the Varieties information is confounded with differences between plots within blocks. Of course, the remaining 20% is confounded with Blocks.

3.2.2 Anova for the Yields

```
#'## Perform an anova
summary(aov(Yield ~ Varieties + Error(Blocks/Plots), data = BIBDWheat.dat))

##
## Error: Blocks
##           Df Sum Sq Mean Sq F value Pr(>F)
## Varieties  5  196.6   39.32   0.582  0.718
## Residuals  4  270.4   67.59
##
## Error: Blocks:Plots
##           Df Sum Sq Mean Sq F value Pr(>F)
## Varieties  5 1156.4  231.29   4.021 0.0163 *
## Residuals 15  862.9   57.53
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3.2.3 Questions

1. What is the value of xefficiency for Varieties when confounded with Plots[Blocks] for this design? Why?
   It is 0.80 because there is only the one value for the canonical efficiency factor between these two sources
2. How many nonzero eigenvalues does $\mathbf{Q}_{\mathrm{V}}\mathbf{Q}_{\mathrm{BP}}\mathbf{Q}_{\mathrm{V}}$ have?
   It has 5 nonzero eigenvalues because there is 5 df of Varieties confounded with Plots[Blocks].

3.3 A design with rows and columns from Williams (2002)

Williams et al. (2002, p.144) provide an example of a tree experiment that investigated differences between 60 provenances of a species of Casuarina tree, these provenances coming from 18 countries; the trees were inoculated prior to planting at two different times. The design used was a split-unit design comprised of four rectangles each of six rows by ten columns; the rectangles are located next to each other so that they are contiguous along the rows. The two inoculation times were randomized to the rectangles (main units). The provenances were randomized to the subunits using a resolved, latinized, row-column design, the rectangles forming replicates of the Provenances. The latinization was by columns and was necessary because differences between Columns (across Reps) was anticipated; it served to avoid multiple occurrences of a provenance in a column. At 30 months, diameter at breast height (Dbh) was measured.

The factor-allocation diagram for the experiment is in Figure 8.

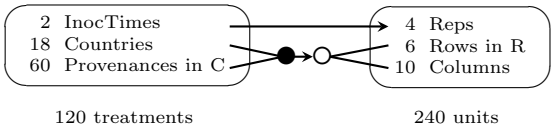


Figure 8: Factor-allocation diagram for the row-and-column design: treatments are allocated to units; the arrows indicates that the allocations are randomized; the two lines leading to the '●' indicate that it is is the combinations of Countries and Provenances that is allocated; the '○' at the end of the lower arrow indicates that a nonorthogonal design is used; the two lines from '○' indicates that the Countries and Provenances are allocated to the combinations of Rows and Columns using the design; Rows in B indicates that the Rows are considered to be nested within Reps for this randomization; R = Reps.

3.3.1 Input the design and check the properties of the design

Use the following R code to input the design and check its properties.

```
#'## Input the design
data("Casuarina.dat")
#'## Check the properties of the design
Casuarina.canon <- designAnatomy(formulae = list(units = ~ (Reps/Rows)*Columns,
                                                 trts  = ~ InocTime*(Countries+Provenances)),
                                 data     = Casuarina.dat)

## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Provenances[Countries]
and Countries are partially aliased in Rows[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Provenances[Countries]
and Countries are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Provenances[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Countries and
Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Countries and
Provenances[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Provenances[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and InocTime#Countries are partially aliased in Rows#Columns[Reps]

summary(Casuarina.canon, which = c("aeff", "eeff", "order", "dforth"))

##
##
## Summary table of the decomposition for units & trts (based on adjusted quantities)
##
##  Source.units       df1 Source.trts                    df2 aefficiency eefficiency order dforthog
##  Reps                 3 InocTime                         1      1.0000      1.0000     1        1
##                         Residual                         2
##  Rows[Reps]          20 Countries                       17      0.0145      0.0018    17        0
##                         Provenances[Countries]           3      0.1622      0.1326     3        0
##  Columns              9 Countries                       9      0.0137      0.0028     9        0
##  Reps#Columns        27 Countries                       17      0.0134      0.0012    17        0
##                         Provenances[Countries]          10      0.2320      0.1596    10        0
##  Rows#Columns[Reps] 180 Countries                       17      0.7611      0.5588    17        0
##                         Provenances[Countries]          42      0.6851      0.3429    42        0
```

```
##                         InocTime#Countries                17      0.6808      0.4735    17        0
##                         InocTime#Provenances[Countries]   42      0.5516      0.2009    42        0
##                         Residual                          62
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source                           df Alias                   In                 aefficiency
##  Provenances[Countries]           17 Countries               Rows[Reps]              1.0000
##  Provenances[Countries]           17 Countries               Reps#Columns            1.0000
##  Provenances[Countries]           17 Countries               Rows#Columns[Reps]      0.0178
##  InocTime#Countries               17 Countries               Rows#Columns[Reps]      0.0001
##  InocTime#Countries               17 Provenances[Countries] Rows#Columns[Reps]      0.0222
##  InocTime#Provenances[Countries] 17 Countries               Rows#Columns[Reps]      0.0222
##  InocTime#Provenances[Countries] 42 Provenances[Countries] Rows#Columns[Reps]      0.0000
##  InocTime#Provenances[Countries] 17 InocTime#Countries      Rows#Columns[Reps]      0.0178
##  eefficiency order dforthog
##       1.0000     1       17
##       1.0000     1       17
##       0.0025    17        0
##       0.0000    17        0
##       0.0042    17        0
##       0.0042    17        0
##       0.0000    42        0
##       0.0025    17        0
##
## The design is not orthogonal
```

Firstly, note that `designAnatomy` has automatically detected that Provenances is nested within Countries, even though Provenances has 60 unique levels: the sources for these two terms are Countries and Provenances[Countries] and these have 17 and 42 degrees of freedom when estimated from Rows # Columns [Reps], respectively. The total of these degrees of freedom is 59, one less than the number of Provenances, as expected.

Secondly, the partial aliasing evident in this design reflects a lack of (structure) balance between the treatment sources within each units source. This is an undesirable, but unavoidable, feature of the design for this experiment.

3.3.2 Questions

1. What is it about the design that makes it resolved for Provenances?

   Each Rep contains all 60 Provenances once and only once, i.e. a complete replicate of the Provenances.

2. What is the disadvantage of allocating InocTimes to Reps?

   There are only two Residual degrees of freedom for testing for the main effect for InocTimes.

3.4 A resolved design for the wheat experiment that is near-A-optimal under a mixed model

Gilmour et al. (1995) provides an example of a wheat experiment for 25 Varieties in which a balanced lattice square design was employed, it being a resolved row-column design.

The factor-allocation diagram for the experiment is in Figure 9.

3.4.1 Input the design and check the properties of the design

The design is available in the Wheat data set in the asremlPlus package (Brien, 2023a). Use the following R code to input the design, plot it and check its properties.

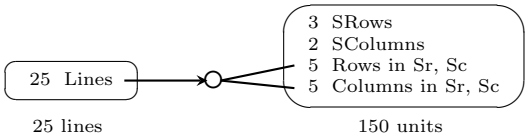
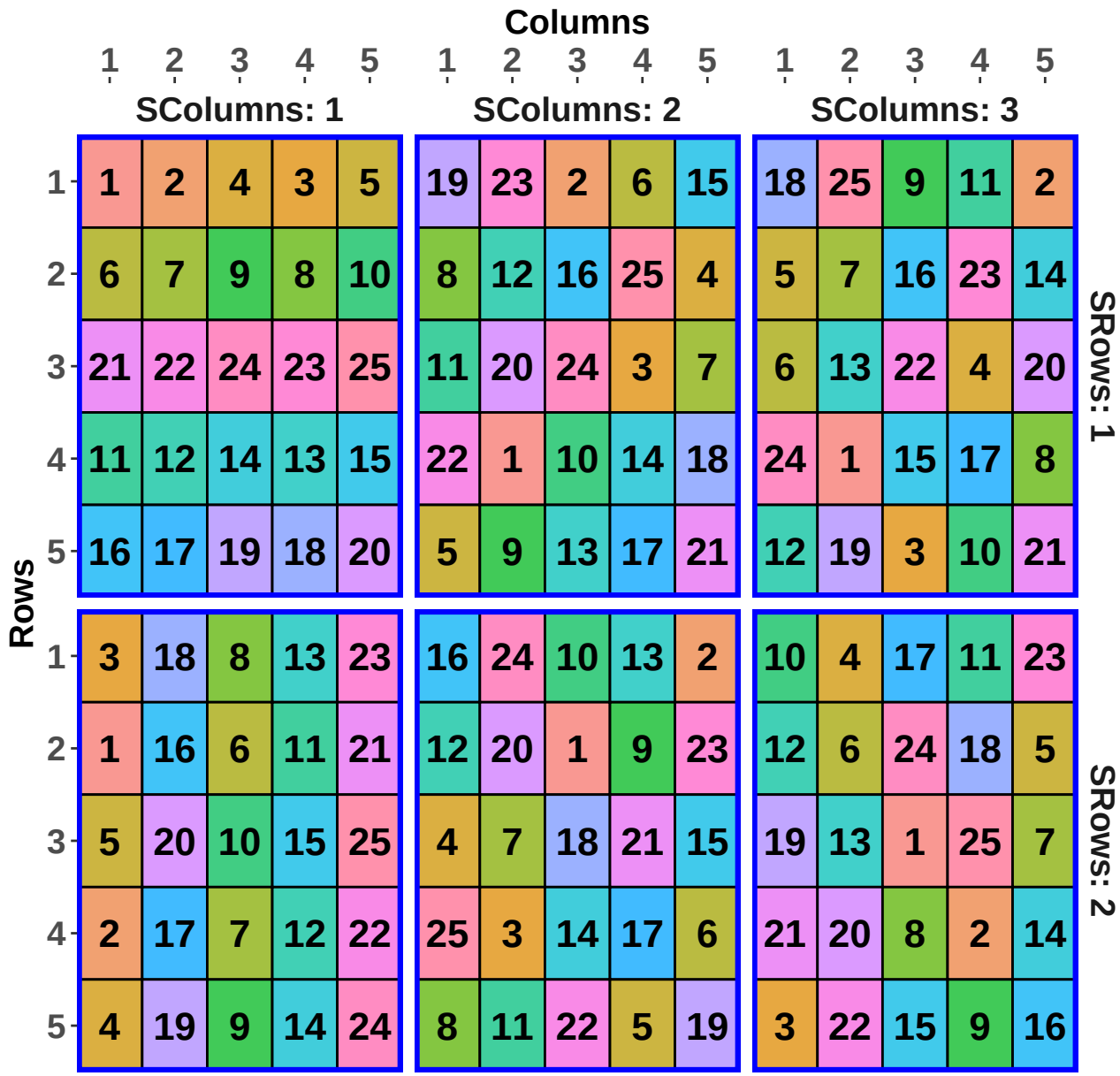


Figure 9: Factor-allocation diagram for the balanced lattice square design: treatments are allocated to units; the arrows indicates that the allocations are randomized; the 'O' at the end of the lower arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicates that the Lines are allocated to the combinations of Rows and Columns using the design; Rows (Columns) in Sr, Sc indicates that the Rows (Columns) are considered to be nested within SRows and SColumns for this randomization; Sr = S(uper)Rows; Sc = S(uper)Columns.

```
#'## Get the design
library(asremlPlus)

## ASReml-R needs to be loaded if the mixed-model functions are to be used.
##
## ASReml-R is available from VSNi.  Please visit http://www.vsni.co.uk/ for more information.

data(Wheat.dat)
latt.lay <- cbind(fac.gen(list(SRows = 2, Rows = 5, SColumns = 3, Columns =5)),
                  Wheat.dat["Variety"])
#'## Plot the design
#+ "LattDesign"
library(scales)
cell.colours <- hue_pal()(25)
designGGPlot(latt.lay, labels = "Variety",
             row.factors = c("SRows", "Rows"), column.factors = c("SColumns", "Columns"),
             colour.values = cell.colours, cellalpha = 0.75, size = 6,
             blockdefinition = cbind(5,5))
```




```
#'## Check the properties of the design
latt.canon <- designAnatomy(formulae = list(units = ~ (SRows:SColumns)/(Rows*Columns),
                                            trts  = ~ Variety),
                            data     = latt.lay)
summary(latt.canon, which.criteria = c("aeff", "order"))
```

```
##
##
## Summary table of the decomposition for units & trts (based on adjusted quantities)
##
##  Source.units                 df1 Source.trts df2 aefficiency order
##  SRows:SColumns                 5
##  Rows[SRows:SColumns]          24 Variety      24      0.1667     1
```

```
##  Columns[SRows:SColumns]       24 Variety      24      0.1667     1
##  Rows#Columns[SRows:SColumns]  96 Variety      24      0.6667     1
##                                   Residual     72
##
## The design is not orthogonal
```

4 Miscellaneous experimental design topics in R

This section includes examples showing the effects of missing values, recognizing pseudoreplication and the use of nested factorials.

Firstly, initialize by loading the libraries that will be used and setting the output width.

```
library(dae, quietly = TRUE)
options(width=100)
```

4.1 An environmental experiment

Suppose an environmental scientist wants to investigate the effect on the biomass of burning areas of natural vegetation. There are available two areas separated by several kilometres for use in the investigation. It is only possible to either burn or not burn an entire area. The area to be burnt is randomly selected and the other area is to be left unburnt as a control. Further, 30 locations in each area are to be randomly sampled and the biomass measured at each location. The factor-allocation diagram for the experiment is in Figure 10.

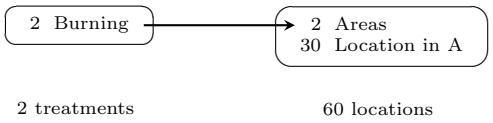


Figure 10: Factor-allocation diagram for the environmental experiment: treatments are allocated to locations; the arrow indicates that the factor Burning is randomized to Areas; Locations in A indicates that the Locations are considered to be nested within Areas; A = Areas.

Obtain the randomized layout for this experiment and check its properties.

```
#'## Obtain the layout
Burn.sys <- cbind(fac.gen(list(Areas=2, Locations=30)),
                  Burn = factor(rep(c("Burn", "NoBurn"), each=30)))
Burn.lay <- designRandomize(allocated         = Burn.sys["Burn"],
                            recipient         = Burn.sys[c("Areas", "Locations")],
                            nested.recipients = list(Locations = "Areas"),
                            seed              = 872159)

#'## plot the design
designGGPlot(Burn.lay, labels = "Burn", row.factors = "Locations", column.factors = "Areas")
```


```
#'## Check its properties
Burn.canon <- designAnatomy(formulae = list(unit = ~Areas/Locations,
                                            trt  = ~Burn),
                            data     = Burn.lay)
summary(Burn.canon)

##
##
## Summary table of the decomposition for unit & trt
##
##  Source.unit      df1 Source.trt df2 aefficiency eefficiency order
##  Areas              1 Burn         1      1.0000      1.0000     1
##  Locations[Areas]  58
```

4.1.1 Questions

1. How is the pseudo-replication involved in this experiment manifested in the skeleton anova table?

   Because (i) Areas and Burn are alongside each other in the anova table, (ii) they both have 1 degree of freedom, and (iii) the single canonical efficiency factor is one, then Areas and Burn are completely confounded. That is, the pseudoreplication has resulted in differences between Areas and between Burns being inextricably mixed up.

2. The randomization-based mixed model for the experiment is Burn | Areas + Areas:Locations. What difficulties do you anticipate in attempting to fit this model? How could the model be modified so that a fit can be obtained? Brien and Demétrio (2009) call models formed by removing terms to enable a fit to be achieved 'models of convenience'. What dangers do you foresee in basing conclusions on the fitted model of convenience?

   There will be a singularity in the model because Areas is confounded with Burn. A fit could be obtained by removing Areas from the random model. The problem is that a test of Burn would then be based on the ratio of variability in Burn differences to an estimate of the variance of Locations-within-Areas variability. This does not include Areas variability and so the denominator is likely to be underestimated; p-values based from this test are likely to be too small and significant differences are more likely to be declared where there are none as compared to when an estimate of Areas variability is included in the denominator of the F-statistic.

4.2 A detergent experiment

Mead et al. (2012) describe an experiment to investigate nine detergent formulations that were compared by washing plates one at a time until they were clean. There were only 3 basins available at any one time and so a BIBD with 12 blocks was used to assign formulations to washing instances. Each basin has a different operator who washed at the same rate at each time of washing. The response is the number of plates washed before the foam disappears.

The treatments involve two bases, four additive amounts and a control; they are:

1. base I + three parts additive
2. base I + two parts additive
3. base I + one part additive
4. base I
5. base II + three parts additive
6. base II + two parts additive
7. base II + one part additive
8. base II
9. Control

The factor-allocation diagram for the experiment is in Figure 11.

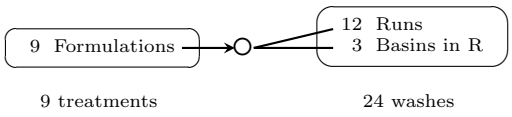


Figure 11: Factor-allocation diagram for the detergent experiment: treatments are allocated to washes; the arrow indicates that the allocation is randomized; the 'O' at the end of the arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicate that the Treatments are allocated to the combinations of Runs and Basins using the design; Basins in R indicates that the Basins are considered to be nested within Runs for this randomization; R = Runs.

The systematic incomplete-block design is shown in Table 2.

4.2.1 Produce the randomized layout for the BIBD and check its properties

Table 2: Systematic balanced incomplete-block design for 9 treatments in blocks of 3

	Basin		
Run	1	2	3
1	1	2	3
2	4	5	6
3	7	8	9
4	1	4	7
5	2	5	8
6	3	6	9
7	1	5	9
8	2	6	7
9	3	4	8
10	1	6	8
11	2	4	9
12	3	5	7

```
b <- 12
k <- 3
t <- 9

#'## Input the systematic design and randomize
BIBD.sys <- cbind(fac.gen(list(Runs = b, Basins = k)),
                  Formulations = factor(c(1:9,
                                          1, 4, 7,
                                          2, 5, 8,
                                          3, 6, 9,
                                          1, 5, 9,
                                          2, 6, 7,
                                          3, 4, 8,
                                          1, 6, 8,
                                          2, 4, 9,
                                          3, 5, 7)))
#'## Randomize the systematic design
BIBD.lay <- designRandomize(allocated         = BIBD.sys["Formulations"],
                            recipient         = BIBD.sys[c("Runs", "Basins")],
                            nested.recipients = list(Basins = "Runs"),
                            seed              = 64686)

#'### Check properties of the BIBD
BIBD.canon <- designAnatomy(formulae = list(wash = ~ Runs/Basins,
                                            form = ~ Formulations),
                            data     = BIBD.lay)
summary(BIBD.canon, which.criteria = c('aeff', 'order'))

##
##
## Summary table of the decomposition for wash & form (based on adjusted quantities)
##
##  Source.wash  df1 Source.form  df2 aefficiency order
##  Runs          11 Formulations   8      0.2500     1
```

```
##                    Residual       3
##  Basins[Runs]   24 Formulations   8      0.7500     1
##                    Residual      16
##
## The design is not orthogonal
```

4.2.2 Add nested factors and check the decomposition using them

```
BIBD.lay <- within(BIBD.lay,
                   {
                     Types <- fac.uselogical(Formulations == "9", labels = c("Control", "New"))
                     Bases <- fac.recast(Formulations,
                                         newlevels = c(rep(c("I", "II"), each = 4), "Control"))
                     Additives <- fac.recast(Formulations,
                                             newlevels = c(rep(c("four", "three", "two", "none"),
                                                               times = 2), "Control"))
                   })

BIBD.nest.canon <- designAnatomy(formulae = list(wash = ~ Runs/Basins,
                                                 form = ~ Types/(Bases*Additives)),
                                 data     = BIBD.lay)
summary(BIBD.nest.canon, which.criteria = c('aeff', 'order'))

##
##
## Summary table of the decomposition for wash & form (based on adjusted quantities)
##
##  Source.wash  df1 Source.form            df2 aefficiency order
##  Runs          11 Types                    1      0.2500     1
##                   Bases[Types]             1      0.2500     1
##                   Additives[Types]         3      0.2500     1
##                   Bases#Additives[Types]   3      0.2500     1
##                   Residual                 3
##  Basins[Runs]   24 Types                    1      0.7500     1
##                   Bases[Types]             1      0.7500     1
##                   Additives[Types]         3      0.7500     1
##                   Bases#Additives[Types]   3      0.7500     1
##                   Residual                16
##
## The design is not orthogonal
```

4.2.3 Leave out Types and try decomposition with Bases and Additives in both orders

```
BIBD.nest2.canon <- designAnatomy(formulae = list(wash = ~ Runs/Basins,
                                                  form = ~ Bases*Additives),
                                  data     = BIBD.lay)
summary(BIBD.nest2.canon, which.criteria = c('aeff', 'order'))

##
##
## Summary table of the decomposition for wash & form (based on adjusted quantities)
```

```
##
##  Source.wash  df1 Source.form     df2 aefficiency order
##  Runs          11 Bases            2      0.2500     1
##                   Additives         3      0.2500     1
##                   Bases#Additives   3      0.2500     1
##                   Residual          3
##  Basins[Runs]  24 Bases            2      0.7500     1
##                   Additives         3      0.7500     1
##                   Bases#Additives   3      0.7500     1
##                   Residual         16
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source    df Alias                    In   aefficiency order
##  Additives 1  Bases                    form      1.0000     1
##  Additives 3  ## Information remaining form      1.0000     1
##
## The design is not orthogonal
```

```
BIBD.nest2.canon <- designAnatomy(formulae = list(wash = ~ Runs/Basins,
                                                  form = ~ Additives*Bases),
                                  data     = BIBD.lay)
summary(BIBD.nest2.canon, which.criteria = c('aeff', 'order'))
```

```
##
##
## Summary table of the decomposition for wash & form (based on adjusted quantities)
##
##  Source.wash  df1 Source.form     df2 aefficiency order
##  Runs          11 Additives        4      0.2500     1
##                   Bases            1      0.2500     1
##                   Additives#Bases  3      0.2500     1
##                   Residual         3
##  Basins[Runs]  24 Additives        4      0.7500     1
##                   Bases            1      0.7500     1
##                   Additives#Bases  3      0.7500     1
##                   Residual        16
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source df Alias                    In   aefficiency order
##  Bases  1  Additives                form      1.0000     1
##  Bases  1  ## Information remaining form      1.0000     1
##
## The design is not orthogonal
```

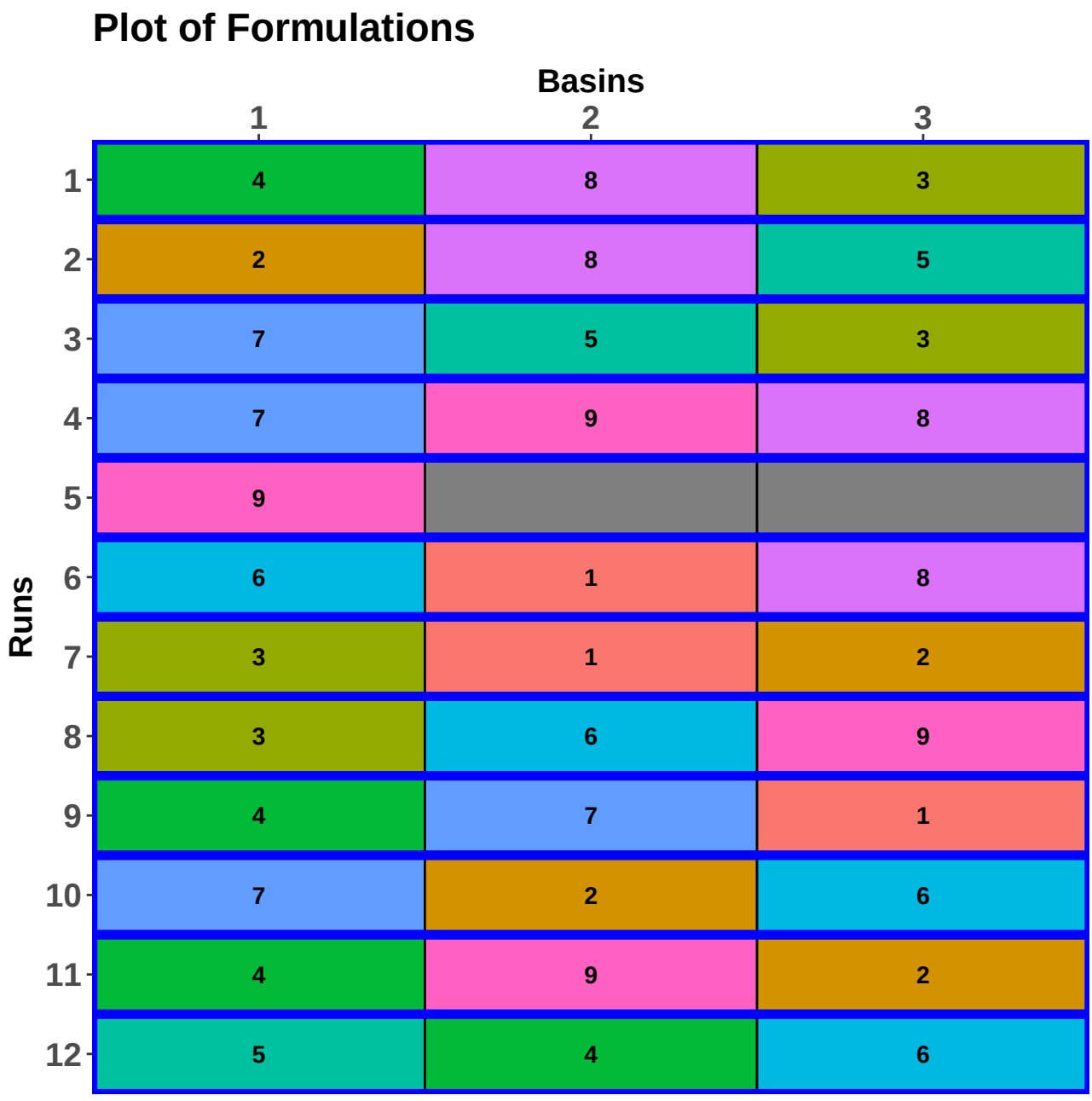
4.2.4 What if two observations are missing?

Two observations that are not the Control are set to missing and the anatomy obtained. The greatest effect is surprisingly on the comparison between the Control and New.

```
#'## Investigate the effect of two-missing observations
#+ "BIBDDet"
```

```
BIBD.Miss.lay <- BIBD.lay
BIBD.Miss.lay$Formulations[c(14,15)] <- NA
designGGPlot(BIBD.Miss.lay, labels = "Formulations",
             row.factors = "Runs", column.factors = "Basins",
             blockdefinition = rbind(c(1,3)))
```

```
## Warning:  Removed 2 rows containing missing values (‘geom_text()‘).
```




```
BIBD.Miss.canon <- designAnatomy(formulae = list(wash = ~ Runs/Basins,
                                                 form = ~ Types/(Bases*Additives)),
                                 data     = na.omit(BIBD.Miss.lay))
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Additives[Types] and
```

```
Types are partially aliased in Runs
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Bases#Additives[Types]
and Bases[Types] are partially aliased in Runs
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Additives[Types] and
Types are partially aliased in Basins[Runs]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Bases#Additives[Types]
and Bases[Types] are partially aliased in Basins[Runs]
```

```
summary(BIBD.Miss.canon, which.criteria = c('aeff', 'order'))
```

4.2.5 Questions

1. What do you conclude about the properties of the design both without and with the nested factors?

   Without the nested factors, the BIBD is balanced. It retains this balance when Formulations is partitioned using the nested factors. This is to be expected with a balanced design because all Formulations contrasts have the same efficiency. The intrablock efficiency factor is 0.75, which is acceptable

2. What is the effect of removing the Types factor?

   The one df for Types is included with the main effect fitted immediately after Types. Clearly the Types factor needs to be separated out before fitting the other factors to remove this arbitrariness in composition of sources.

3. What is the advantage of using nested factors for this experiment?

   It enables the main effects and interactions of Bases and Additives to be explored.

4. Is there any reason to think that a row-column design might be better than a block design for this experiment?

There would be if the same three operators are used for each Run, and there is reason to believe that systematic differences between the operators. A row-column design would reduce the influence of these differences on the precision of the experiment.

4.3 An experiment to investigate the effects of spraying Sultana grapes

Clingeleffer et al. (1977) report an experiment to investigate the effects of tractor speed and spray pressure on the quality of dried sultanas. The response was the lightness of the dried sultanas which is measured using a Hunterlab D25 L colour difference meter. Lighter sultanas are considered to be of better quality and these will have a higher lightness measurement (L). There were three tractor speeds and two spray pressures resulting in 6 treatment combinations which were applied to 6 plots, each consisting of 12 vines, using a randomized complete-block design with three blocks. However, these 6 treatment combinations resulted in only 4 rates of spray application as indicated in the following table.

Table 3: Application rates for the sprayer experiment

	Tractor speed (km hr^{-1})		
Pressure (kPa)	3.6	2.6	1.8
140	2090	2930	4120
330	2930	4120	5770

That is, there are 4 different rates of application, two of which have different combinations of Tractor speed and Spray pressure. So, a factor, Rates, with four levels is set up to compare the means of the four rates and then separate nested factors for each rate are generated.

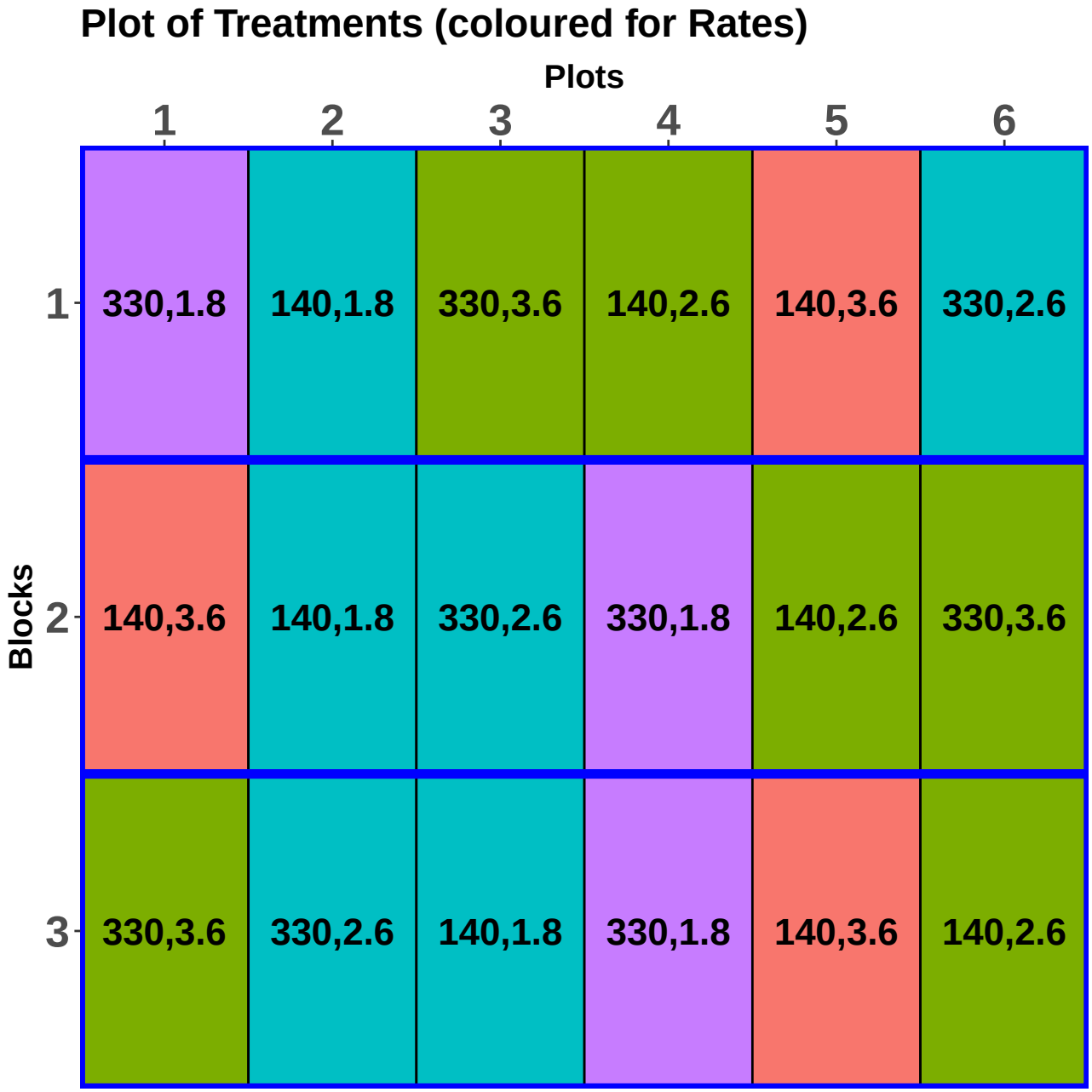
We set up the RCBD for Speed and Pressure then derive the Rate factors.

```
b <- 3
t <- 6
#'## Construct a systematic layout
RCBD.sys <- cbind(fac.gen(generate = list(Blocks=b, Plots=t)),
                  fac.gen(generate = list(Pressure = c("140", "330"),
                                          Speed = c("3.6", "2.6", "1.8")), times = b))

#'## Obtain the randomized layout
RCBD.lay <- designRandomize(allocated         = RCBD.sys[c("Pressure", "Speed")],
                            recipient         = RCBD.sys[c("Blocks", "Plots")],
                            nested.recipients = list(Plots = "Blocks"),
                            seed              = 353441)
#'## Add nested factors
RCBD.lay <- within(RCBD.lay,
                   {
                     Treatments <- fac.combine(list(Pressure, Speed), combine.levels = TRUE)
                     Rates <- fac.recast(Treatments,
                                         newlevels = c("2090", "2930", "4120",
                                                       "2930", "4120", "5770"))
                   })
RCBD.lay <- with(RCBD.lay, cbind(RCBD.lay,
                                 fac.multinested(nesting.fac = Rates,
                                                 nested.fac  = Treatments,
                                                 fac.prefix  = "Rate")))

#'## Output the layout
```

```
RCBD.lay
```

```
##    Blocks Plots Pressure Speed Rates Treatments Rate2090 Rate2930 Rate4120 Rate5770
## 1       1     1      330   1.8  5770    330,1.8     rest     rest     rest  330,1.8
## 2       1     2      140   1.8  4120    140,1.8     rest     rest  140,1.8     rest
## 3       1     3      330   3.6  2930    330,3.6     rest  330,3.6     rest     rest
## 4       1     4      140   2.6  2930    140,2.6     rest  140,2.6     rest     rest
## 5       1     5      140   3.6  2090    140,3.6  140,3.6     rest     rest     rest
## 6       1     6      330   2.6  4120    330,2.6     rest     rest  330,2.6     rest
## 7       2     1      140   3.6  2090    140,3.6  140,3.6     rest     rest     rest
## 8       2     2      140   1.8  4120    140,1.8     rest     rest  140,1.8     rest
## 9       2     3      330   2.6  4120    330,2.6     rest     rest  330,2.6     rest
## 10      2     4      330   1.8  5770    330,1.8     rest     rest     rest  330,1.8
## 11      2     5      140   2.6  2930    140,2.6     rest  140,2.6     rest     rest
## 12      2     6      330   3.6  2930    330,3.6     rest  330,3.6     rest     rest
## 13      3     1      330   3.6  2930    330,3.6     rest  330,3.6     rest     rest
## 14      3     2      330   2.6  4120    330,2.6     rest     rest  330,2.6     rest
## 15      3     3      140   1.8  4120    140,1.8     rest     rest  140,1.8     rest
## 16      3     4      330   1.8  5770    330,1.8     rest     rest     rest  330,1.8
## 17      3     5      140   3.6  2090    140,3.6  140,3.6     rest     rest     rest
## 18      3     6      140   2.6  2930    140,2.6     rest  140,2.6     rest     rest
```

```
#'## Plot the layout
#+ "RCBDSpray_v1"
designGGPlot(RCBD.lay, labels = "Treatments",
             cellfillcolour.column = "Rates",
             row.factors = "Blocks", column.factors = "Plots",
             axis.text.size = 20, size = 6,
             title = "Plot of Treatments (coloured for Rates)",
             blockdefinition = cbind(1,t))
```



Now check the properties of the design with the nested factors.

```
RCBD.canon <- designAnatomy(formulae = list(plots = ~ Blocks/Plots,
                                            trts  = ~ Rates/(Rate2090 + Rate2930 + Rate4120 +
                                                              Rate5770)),
                            data     = RCBD.lay)

## Warning in pstructure.formula(formulae[[ktier]], keep.order = keep.order, :  Rates:Rate2090
is aliased with previous terms in the formula and has been removed
## Warning in pstructure.formula(formulae[[ktier]], keep.order = keep.order, :  Rates:Rate5770
is aliased with previous terms in the formula and has been removed

summary(RCBD.canon, which.criteria = "aeff")
```

```
##
##
## Summary table of the decomposition for plots & trts (based on adjusted quantities)
##
##  Source.plots  df1 Source.trts     df2 aefficiency
##  Blocks          2
##  Plots[Blocks]  15 Rates             3      1.0000
##                    Rate2930[Rates]   1      1.0000
##                    Rate4120[Rates]   1      1.0000
##                    Residual         10
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source         df Alias      In   aefficiency
##  Rates:Rate2090 3  Rates      trts      1.0000
##  Rates:Rate2090 0  ## Aliased trts      1.0000
##  Rates:Rate5770 3  Rates      trts      1.0000
##  Rates:Rate5770 0  ## Aliased trts      1.0000
```

4.3.1 Questions

1. What is the prior allocation model for this design?

   The initial allocation mixed model is Pressure + Speed + Pressure:Speed | Blocks + Blocks:Plots. The fixed model is reparameterized to be based on Rates terms: Rates + Rates:Rates2930 + Rates:Rates4120 | Blocks + Blocks:Plots. The fixed model can also be specified simply as Rates + Rates2930 + Rates4120.

2. How does the prior allocation model differ from the randomization model for this design?

   Only in its parameterization of the fixed model, although Blocks might also be moved to the fixed model.

3. Why are terms involving Rate2090 and Rate5770 not included in the prior allocation model?

   Because there is only one combination of Pressure and Speed for each of these Rates so that, as shown in the Table of aliasing accompanying the Summary table for the anatomy, both Rate2090 and Rate5770 are aliased with Rates.

4.4 A Control treatment for an incomplete-block design

An incomplete-block design for 6 treatments in 6 blocks of size 4 is obtained from Cochran and Cox (1957, p. 379).

```
b <- 6
k <- 4
t <- 6

#'## Input the systematic design and randomize
PBIBD.sys <- cbind(fac.gen(list(Blocks = b, Units = k)),
                   Treatments = factor(c(1,4,2,5,
                                         2,5,3,6,
                                         3,6,1,4,
                                         4,1,5,2,
                                         5,2,6,3,
                                         6,3,4,1)))
```

Randomize the design and check its properties

```
#'### Randomize design according to the plots structure
PBIBD.lay <- designRandomize(allocated         = PBIBD.sys["Treatments"],
                             recipient         = PBIBD.sys[c("Blocks", "Units")],
                             nested.recipients = list(Units = "Blocks"),
                             seed              = 65460)
PBIBD.lay
```

```
##    Blocks Units Treatments
## 1       1     1          1
## 2       1     2          3
## 3       1     3          4
## 4       1     4          6
## 5       2     1          1
## 6       2     2          2
## 7       2     3          5
## 8       2     4          4
## 9       3     1          4
## 10      3     2          1
## 11      3     3          6
## 12      3     4          3
## 13      4     1          2
## 14      4     2          3
## 15      4     3          6
## 16      4     4          5
## 17      5     1          1
## 18      5     2          4
## 19      5     3          2
## 20      5     4          5
## 21      6     1          2
## 22      6     2          5
## 23      6     3          3
## 24      6     4          6
```

```
#'### Check properties of the od layout
PBIBD.canon <- designAnatomy(formulae = list(plots = ~ Blocks/Units,
                                             trts  = ~ Treatments),
                             data      = PBIBD.lay)
summary(PBIBD.canon, which.criteria = c('aeff', 'xeff', 'eeff','order', 'dforth'))
```

```
##
##
## Summary table of the decomposition for plots & trts (based on adjusted quantities)
##
##  Source.plots  df1 Source.trts df2 aefficiency xefficiency eefficiency order dforthog
##  Blocks          5 Treatments    2      0.2500      0.2500      0.2500     1        0
##                    Residual      3
##  Units[Blocks]  18 Treatments    5      0.8824      1.0000      0.7500     2        3
##                    Residual     13
##
## The design is not orthogonal
```

Investigate the effect of designating a treatment to be a Control and including a Control factor in the fixed model. It is noted that, in this case at least, it does not matter which treatment is designated to be the control.

```
#'## Investigate a Control contrast (say treatment 1) for the od design
PBIBD.lay$Control <- with(PBIBD.lay, fac.uselogical(Treatments == "A",
                                                    labels = c("Control", "rest")))
PBIBD.canon <- designAnatomy(formulae = list(unit = ~ Blocks/Units,
                                             trt  = ~ Control + Treatments),
                             data     = PBIBD.lay)

## Error in svd.x$v[, nonzero.x] %*% geninv.x[nonzero.x, nonzero.x]:  non-conformable arguments

summary(PBIBD.canon, which.criteria = c('aeff', 'xeff', 'eeff','order', 'dforth'))

##
##
## Summary table of the decomposition for plots & trts (based on adjusted quantities)
##
##  Source.plots  df1 Source.trts df2 aefficiency xefficiency eefficiency order dforthog
##  Blocks          5 Treatments    2      0.2500      0.2500      0.2500     1        0
##                    Residual      3
##  Units[Blocks]  18 Treatments    5      0.8824      1.0000      0.7500     2        3
##                    Residual     13
##
## The design is not orthogonal

#'### Try other treatments
PBIBD.lay$Control <- with(PBIBD.lay, fac.uselogical(Treatments == "C",
                                                    labels = c("Control", "rest")))
#Rerun the designAnatomy and summary functions
```

1. Why must the Control source be balanced?

   Because it has a single degree of freedom and so there can only be one value for the single efficiency factor.

4.5 The Casuarina experiment (continued)

In Section 3.3 an exploration was made of the properties of the split-unit design for an experiment to investigate the differences between 60 provenances of a species of Casuarina tree, these provenances coming from 18 countries; the trees were inoculated prior to planting at two different times.

The experiment involves nested factors in that the provenances came from 12 countries so that the factor Provenances is nested within Countries. Here we investigate a model that has separate terms for each country that model differences between provenances from each country. Use the `dae` function `fac.multinested` to generate the individual nested factors for each country.

```
#'## Input the design
data(Casuarina.dat)
#'## Add the nested factors
Casuarina.dat <- cbind(Casuarina.dat,
                       with(Casuarina.dat, fac.multinested(nesting.fac = Countries,
                                                           nested.fac = Provenances,
                                                           fac.prefix = "Prov_")))
```

This example has two difficulties that need to be dealt with. Firstly, a number of Countries contribute only one Provenance and terms for differences amongst provenances from those countries are superfluous. Secondly, because of the large number of terms and considerable nonothogonality in the design, it is difficult to get a full decomposition. To overcome this, the following measures are taken:

- Leave out nested terms for countries with only a single provenance;
- Reduce the tolerances on testing for idempotency using the function `set.daeTolerance`;
- Do not attempt to partition the InocTimes#Provenances[Countries] interaction.

```
#'## Produce a list of Countries that have one than Provenance and construct the trts formula
fac.names <- paste0("Prov_", levels(Casuarina.dat$Countries))
no.prov <- unlist(lapply(Casuarina.dat[fac.names], function(fac) length(levels(fac[1]))-1))
(multProv <- names(no.prov[no.prov > 1]))
```

```
##  [1] "Prov_Australia"   "Prov_China"       "Prov_Egypt"       "Prov_Fiji"        "Prov_India"
##  [6] "Prov_Kenya"       "Prov_Malaysia"    "Prov_Phillipines" "Prov_SolomomIs"   "Prov_SriLanka"
## [11] "Prov_Thailand"    "Prov_Vanuatu"     "Prov_Vietnam"
```

```
trts.form <- as.formula(paste0("~ Countries/(",
                               paste0(multProv, collapse = "+"),
                               ")+InocTime/Countries/Provenances"))
(trts.form)
```

```
## ~Countries/(Prov_Australia + Prov_China + Prov_Egypt + Prov_Fiji +
##     Prov_India + Prov_Kenya + Prov_Malaysia + Prov_Phillipines +
##     Prov_SolomomIs + Prov_SriLanka + Prov_Thailand + Prov_Vanuatu +
##     Prov_Vietnam) + InocTime/Countries/Provenances
```

```
#'## Check the properties of the design
set.daeTolerance(1e-05)
Casuarina.canon <- designAnatomy(formulae   = list(units = ~ (Reps/Rows)*Columns,
                                                   trts  = trts.form),
                                 keep.order = TRUE,
                                 data       = Casuarina.dat)
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Australia[Countries]
and Countries are partially aliased in Rows[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Australia[Countries]
and Countries are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_China[Countries]
and Countries are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_China[Countries]
and Prov_Australia[Countries] are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Egypt[Countries]
and Countries are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Fiji[Countries] and
Countries are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Fiji[Countries] and
Prov_Australia[Countries] are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Fiji[Countries] and
Prov_Egypt[Countries] are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Countries are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Prov_Australia[Countries] are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Prov_China[Countries] are partially aliased in Reps#Columns
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Prov_Egypt[Countries] are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Prov_Fiji[Countries] are partially aliased in Reps#Columns
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Australia[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_China[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_China[Countries]
and Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Egypt[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Egypt[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Fiji[Countries] and
Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Fiji[Countries] and
Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Fiji[Countries] and
Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_India[Countries]
and Prov_Fiji[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Kenya[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Kenya[Countries]
and Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Kenya[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Kenya[Countries]
and Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Kenya[Countries]
and Prov_Fiji[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Kenya[Countries]
and Prov_India[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Malaysia[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Malaysia[Countries]
and Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Malaysia[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Malaysia[Countries]
and Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Malaysia[Countries]
and Prov_Fiji[Countries] are partially aliased in Rows#Columns[Reps]
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Malaysia[Countries]
and Prov_India[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Malaysia[Countries]
and Prov_Kenya[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Phillipines[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Phillipines[Countries]
and Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Phillipines[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Phillipines[Countries]
and Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Phillipines[Countries]
and Prov_India[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Phillipines[Countries]
and Prov_Kenya[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Phillipines[Countries]
and Prov_Malaysia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_SolomomIs[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_SolomomIs[Countries]
and Prov_Kenya[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_SolomomIs[Countries]
and Prov_Malaysia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_SriLanka[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_SriLanka[Countries]
and Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_SriLanka[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_SriLanka[Countries]
and Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_SriLanka[Countries]
and Prov_Kenya[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_SriLanka[Countries]
and Prov_Malaysia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Thailand[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Thailand[Countries]
and Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Thailand[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Thailand[Countries]
and Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Thailand[Countries]
and Prov_Fiji[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Thailand[Countries]
and Prov_India[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Thailand[Countries]
and Prov_Kenya[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Thailand[Countries]
and Prov_Malaysia[Countries] are partially aliased in Rows#Columns[Reps]
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Thailand[Countries]
and Prov_Phillipines[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vanuatu[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vanuatu[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vanuatu[Countries]
and Prov_India[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vanuatu[Countries]
and Prov_Kenya[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vanuatu[Countries]
and Prov_Malaysia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vanuatu[Countries]
and Prov_SriLanka[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Prov_India[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Prov_Kenya[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Prov_Malaysia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Prov_Phillipines[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Prov_SriLanka[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Prov_Vietnam[Countries]
and Prov_Thailand[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_Fiji[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_India[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_Kenya[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_Malaysia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_Phillipines[Countries] are partially aliased in Rows#Columns[Reps]
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_SolomomIs[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_SriLanka[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_Thailand[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_Vanuatu[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Countries#InocTime and
Prov_Vietnam[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Countries are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_Australia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_China[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_Egypt[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_Fiji[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_India[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_Kenya[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_Malaysia[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_Phillipines[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_SolomomIs[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_SriLanka[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_Thailand[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_Vanuatu[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Prov_Vietnam[Countries] are partially aliased in Rows#Columns[Reps]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): InocTime#Provenances[Countries]
and Countries#InocTime are partially aliased in Rows#Columns[Reps]
```

```
summary(Casuarina.canon, which = c("aeff", "eeff", "order", "dforth"))
```

```
##
##
## Summary table of the decomposition for units & trts (based on adjusted quantities)
##
##  Source.units       df1 Source.trts                    df2 aefficiency eefficiency order dforthog
##  Reps                 3 InocTime                         1      1.0000      1.0000     1        1
##                         Residual                         2
##  Rows[Reps]          20 Countries                       17      0.0145      0.0018    17        0
##                         Prov_Australia[Countries]        3      0.0001      0.0000     3        0
##  Columns              9 Countries                       9      0.0137      0.0028     9        0
##  Reps#Columns        27 Countries                      17      0.0134      0.0012    17        0
```

```
##                          Prov_Australia[Countries]          3      0.0522      0.0350     3         0
##                          Prov_China[Countries]              1      0.0318      0.0318     1         0
##                          Prov_Egypt[Countries]              2      0.0044      0.0023     2         0
##                          Prov_Fiji[Countries]               2      0.0041      0.0021     2         0
##                          Prov_India[Countries]              2      0.0705      0.0566     2         0
##  Rows#Columns[Reps] 180  Countries                         17      0.7611      0.5588    17         0
##                          Prov_Australia[Countries]          3      0.7259      0.6874     3         0
##                          Prov_China[Countries]              2      0.7260      0.6771     2         0
##                          Prov_Egypt[Countries]              2      0.7346      0.7309     2         0
##                          Prov_Fiji[Countries]               2      0.7314      0.6754     2         0
##                          Prov_India[Countries]              5      0.7097      0.6231     5         0
##                          Prov_Kenya[Countries]              7      0.7128      0.6269     7         0
##                          Prov_Malaysia[Countries]           8      0.7120      0.5745     8         0
##                          Prov_Phillipines[Countries]        2      0.6736      0.6704     2         0
##                          Prov_SolomomIs[Countries]          1      0.6838      0.6838     1         0
##                          Prov_SriLanka[Countries]           2      0.7220      0.6759     2         0
##                          Prov_Thailand[Countries]           3      0.7069      0.6701     3         0
##                          Prov_Vanuatu[Countries]            1      0.7297      0.7297     1         0
##                          Prov_Vietnam[Countries]            4      0.6975      0.6281     4         0
##                          Countries#InocTime                17      0.6808      0.4735    17         0
##                          InocTime#Provenances[Countries]   42      0.5516      0.2009    42         0
##                          Residual                          62
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source                          df Alias                          In                 aefficiency
##  Prov_Australia[Countries]        3 Countries                      Rows[Reps]              0.9251
##  Prov_Australia[Countries]        3 Countries                      Reps#Columns            0.5010
##  Prov_China[Countries]            2 Countries                      Reps#Columns            0.6772
##  Prov_China[Countries]            2 Prov_Australia[Countries]      Reps#Columns            0.0597
##  Prov_Egypt[Countries]            2 Countries                      Reps#Columns            0.7933
##  Prov_Fiji[Countries]             2 Countries                      Reps#Columns            0.4978
##  Prov_Fiji[Countries]             2 Prov_Australia[Countries]      Reps#Columns            0.0028
##  Prov_Fiji[Countries]             2 Prov_Egypt[Countries]          Reps#Columns            0.0645
##  Prov_India[Countries]            5 Countries                      Reps#Columns            0.3421
##  Prov_India[Countries]            3 Prov_Australia[Countries]      Reps#Columns            0.1025
##  Prov_India[Countries]            2 Prov_China[Countries]          Reps#Columns            0.0613
##  Prov_India[Countries]            2 Prov_Egypt[Countries]          Reps#Columns            0.0173
##  Prov_India[Countries]            2 Prov_Fiji[Countries]           Reps#Columns            0.0321
##  Prov_Australia[Countries]        3 Countries                      Rows#Columns[Reps]      0.0161
##  Prov_China[Countries]            2 Countries                      Rows#Columns[Reps]      0.0178
##  Prov_China[Countries]            2 Prov_Australia[Countries]      Rows#Columns[Reps]      0.0003
##  Prov_Egypt[Countries]            2 Countries                      Rows#Columns[Reps]      0.0245
##  Prov_Egypt[Countries]            2 Prov_China[Countries]          Rows#Columns[Reps]      0.0028
##  Prov_Fiji[Countries]             2 Countries                      Rows#Columns[Reps]      0.0110
##  Prov_Fiji[Countries]             2 Prov_Australia[Countries]      Rows#Columns[Reps]      0.0007
##  Prov_Fiji[Countries]             2 Prov_Egypt[Countries]          Rows#Columns[Reps]      0.0005
##  Prov_India[Countries]            5 Countries                      Rows#Columns[Reps]      0.0115
##  Prov_India[Countries]            3 Prov_Australia[Countries]      Rows#Columns[Reps]      0.0040
##  Prov_India[Countries]            2 Prov_China[Countries]          Rows#Columns[Reps]      0.0036
##  Prov_India[Countries]            2 Prov_Egypt[Countries]          Rows#Columns[Reps]      0.0014
##  Prov_India[Countries]            2 Prov_Fiji[Countries]           Rows#Columns[Reps]      0.0042
##  Prov_Kenya[Countries]            7 Countries                      Rows#Columns[Reps]      0.0083
```

```
##  Prov_Kenya[Countries]           3 Prov_Australia[Countries]   Rows#Columns[Reps]      0.0102
##  Prov_Kenya[Countries]           2 Prov_China[Countries]       Rows#Columns[Reps]      0.0082
##  Prov_Kenya[Countries]           2 Prov_Egypt[Countries]       Rows#Columns[Reps]      0.0065
##  Prov_Kenya[Countries]           2 Prov_Fiji[Countries]        Rows#Columns[Reps]      0.0035
##  Prov_Kenya[Countries]           5 Prov_India[Countries]       Rows#Columns[Reps]      0.0015
##  Prov_Malaysia[Countries]        8 Countries                   Rows#Columns[Reps]      0.0068
##  Prov_Malaysia[Countries]        3 Prov_Australia[Countries]   Rows#Columns[Reps]      0.0058
##  Prov_Malaysia[Countries]        2 Prov_China[Countries]       Rows#Columns[Reps]      0.0093
##  Prov_Malaysia[Countries]        2 Prov_Egypt[Countries]       Rows#Columns[Reps]      0.0079
##  Prov_Malaysia[Countries]        2 Prov_Fiji[Countries]        Rows#Columns[Reps]      0.0088
##  Prov_Malaysia[Countries]        5 Prov_India[Countries]       Rows#Columns[Reps]      0.0077
##  Prov_Malaysia[Countries]        7 Prov_Kenya[Countries]       Rows#Columns[Reps]      0.0005
##  Prov_Phillipines[Countries]     2 Countries                   Rows#Columns[Reps]      0.0199
##  Prov_Phillipines[Countries]     2 Prov_Australia[Countries]   Rows#Columns[Reps]      0.0018
##  Prov_Phillipines[Countries]     2 Prov_China[Countries]       Rows#Columns[Reps]      0.0033
##  Prov_Phillipines[Countries]     2 Prov_Egypt[Countries]       Rows#Columns[Reps]      0.0017
##  Prov_Phillipines[Countries]     2 Prov_India[Countries]       Rows#Columns[Reps]      0.0116
##  Prov_Phillipines[Countries]     2 Prov_Kenya[Countries]       Rows#Columns[Reps]      0.0030
##  Prov_Phillipines[Countries]     2 Prov_Malaysia[Countries]    Rows#Columns[Reps]      0.0090
##  Prov_SolomomIs[Countries]       1 Countries                   Rows#Columns[Reps]      0.0244
##  Prov_SolomomIs[Countries]       1 Prov_Kenya[Countries]       Rows#Columns[Reps]      0.0103
##  Prov_SolomomIs[Countries]       1 Prov_Malaysia[Countries]    Rows#Columns[Reps]      0.0108
##  Prov_SriLanka[Countries]        2 Countries                   Rows#Columns[Reps]      0.0192
##  Prov_SriLanka[Countries]        2 Prov_Australia[Countries]   Rows#Columns[Reps]      0.0020
##  Prov_SriLanka[Countries]        2 Prov_China[Countries]       Rows#Columns[Reps]      0.0062
##  Prov_SriLanka[Countries]        2 Prov_Egypt[Countries]       Rows#Columns[Reps]      0.0017
##  Prov_SriLanka[Countries]        2 Prov_Kenya[Countries]       Rows#Columns[Reps]      0.0079
##  Prov_SriLanka[Countries]        2 Prov_Malaysia[Countries]    Rows#Columns[Reps]      0.0027
##  Prov_Thailand[Countries]        3 Countries                   Rows#Columns[Reps]      0.0109
##  Prov_Thailand[Countries]        3 Prov_Australia[Countries]   Rows#Columns[Reps]      0.0000
##  Prov_Thailand[Countries]        2 Prov_China[Countries]       Rows#Columns[Reps]      0.0003
##  Prov_Thailand[Countries]        2 Prov_Egypt[Countries]       Rows#Columns[Reps]      0.0024
##  Prov_Thailand[Countries]        2 Prov_Fiji[Countries]        Rows#Columns[Reps]      0.0065
##  Prov_Thailand[Countries]        3 Prov_India[Countries]       Rows#Columns[Reps]      0.0014
##  Prov_Thailand[Countries]        3 Prov_Kenya[Countries]       Rows#Columns[Reps]      0.0059
##  Prov_Thailand[Countries]        3 Prov_Malaysia[Countries]    Rows#Columns[Reps]      0.0021
##  Prov_Thailand[Countries]        2 Prov_Phillipines[Countries] Rows#Columns[Reps]      0.0019
##  Prov_Vanuatu[Countries]         1 Countries                   Rows#Columns[Reps]      0.0185
##  Prov_Vanuatu[Countries]         1 Prov_China[Countries]       Rows#Columns[Reps]      0.0107
##  Prov_Vanuatu[Countries]         1 Prov_India[Countries]       Rows#Columns[Reps]      0.0070
##  Prov_Vanuatu[Countries]         1 Prov_Kenya[Countries]       Rows#Columns[Reps]      0.0103
##  Prov_Vanuatu[Countries]         1 Prov_Malaysia[Countries]    Rows#Columns[Reps]      0.0044
##  Prov_Vanuatu[Countries]         1 Prov_SriLanka[Countries]    Rows#Columns[Reps]      0.0072
##  Prov_Vietnam[Countries]         4 Countries                   Rows#Columns[Reps]      0.0144
##  Prov_Vietnam[Countries]         3 Prov_Australia[Countries]   Rows#Columns[Reps]      0.0021
##  Prov_Vietnam[Countries]         2 Prov_China[Countries]       Rows#Columns[Reps]      0.0028
##  Prov_Vietnam[Countries]         2 Prov_Egypt[Countries]       Rows#Columns[Reps]      0.0025
##  Prov_Vietnam[Countries]         4 Prov_India[Countries]       Rows#Columns[Reps]      0.0017
##  Prov_Vietnam[Countries]         4 Prov_Kenya[Countries]       Rows#Columns[Reps]      0.0031
##  Prov_Vietnam[Countries]         4 Prov_Malaysia[Countries]    Rows#Columns[Reps]      0.0019
##  Prov_Vietnam[Countries]         2 Prov_Phillipines[Countries] Rows#Columns[Reps]      0.0061
##  Prov_Vietnam[Countries]         2 Prov_SriLanka[Countries]    Rows#Columns[Reps]      0.0080
##  Prov_Vietnam[Countries]         3 Prov_Thailand[Countries]    Rows#Columns[Reps]      0.0005
```

```
##  Countries#InocTime              17 Countries                    Rows#Columns[Reps]      0.0001
##  Countries#InocTime               3 Prov_Australia[Countries]    Rows#Columns[Reps]      0.0147
##  Countries#InocTime               2 Prov_China[Countries]        Rows#Columns[Reps]      0.0186
##  Countries#InocTime               2 Prov_Egypt[Countries]        Rows#Columns[Reps]      0.0182
##  Countries#InocTime               2 Prov_Fiji[Countries]         Rows#Columns[Reps]      0.0085
##  Countries#InocTime               5 Prov_India[Countries]        Rows#Columns[Reps]      0.0114
##  Countries#InocTime               7 Prov_Kenya[Countries]        Rows#Columns[Reps]      0.0095
##  Countries#InocTime               8 Prov_Malaysia[Countries]     Rows#Columns[Reps]      0.0100
##  Countries#InocTime               2 Prov_Phillipines[Countries] Rows#Columns[Reps]      0.0263
##  Countries#InocTime               1 Prov_SolomomIs[Countries]    Rows#Columns[Reps]      0.0198
##  Countries#InocTime               2 Prov_SriLanka[Countries]     Rows#Columns[Reps]      0.0126
##  Countries#InocTime               3 Prov_Thailand[Countries]     Rows#Columns[Reps]      0.0211
##  Countries#InocTime               1 Prov_Vanuatu[Countries]      Rows#Columns[Reps]      0.0099
##  Countries#InocTime               4 Prov_Vietnam[Countries]      Rows#Columns[Reps]      0.0162
##  InocTime#Provenances[Countries] 17 Countries                    Rows#Columns[Reps]      0.0222
##  InocTime#Provenances[Countries]  3 Prov_Australia[Countries]    Rows#Columns[Reps]      0.0647
##  InocTime#Provenances[Countries]  2 Prov_China[Countries]        Rows#Columns[Reps]      0.0604
##  InocTime#Provenances[Countries]  2 Prov_Egypt[Countries]        Rows#Columns[Reps]      0.0636
##  InocTime#Provenances[Countries]  2 Prov_Fiji[Countries]         Rows#Columns[Reps]      0.0779
##  InocTime#Provenances[Countries]  5 Prov_India[Countries]        Rows#Columns[Reps]      0.0693
##  InocTime#Provenances[Countries]  7 Prov_Kenya[Countries]        Rows#Columns[Reps]      0.0528
##  InocTime#Provenances[Countries]  8 Prov_Malaysia[Countries]     Rows#Columns[Reps]      0.0488
##  InocTime#Provenances[Countries]  2 Prov_Phillipines[Countries] Rows#Columns[Reps]      0.0750
##  InocTime#Provenances[Countries]  1 Prov_SolomomIs[Countries]    Rows#Columns[Reps]      0.0579
##  InocTime#Provenances[Countries]  2 Prov_SriLanka[Countries]     Rows#Columns[Reps]      0.0502
##  InocTime#Provenances[Countries]  3 Prov_Thailand[Countries]     Rows#Columns[Reps]      0.0720
##  InocTime#Provenances[Countries]  1 Prov_Vanuatu[Countries]      Rows#Columns[Reps]      0.0442
##  InocTime#Provenances[Countries]  4 Prov_Vietnam[Countries]      Rows#Columns[Reps]      0.0527
##  InocTime#Provenances[Countries] 17 Countries#InocTime           Rows#Columns[Reps]      0.0178
##  eefficiency order dforthog
##       0.8435     3        0
##       0.3667     3        0
##       0.5119     2        1
##       0.0349     2        0
##       0.6920     2        0
##       0.3561     2        0
##       0.0014     2        0
##       0.0514     2        0
##       0.1666     5        0
##       0.0708     3        0
##       0.0356     2        0
##       0.0092     2        0
##       0.0174     2        0
##       0.0113     3        0
##       0.0120     2        0
##       0.0002     2        0
##       0.0229     2        0
##       0.0020     2        0
##       0.0063     2        0
##       0.0004     2        0
##       0.0002     2        0
##       0.0040     5        0
##       0.0018     3        0
```

```
##       0.0021     2        0
##       0.0008     2        0
##       0.0026     2        0
##       0.0025     7        0
##       0.0059     3        0
##       0.0059     2        0
##       0.0043     2        0
##       0.0023     2        0
##       0.0004     5        0
##       0.0017     8        0
##       0.0033     3        0
##       0.0063     2        0
##       0.0058     2        0
##       0.0066     2        0
##       0.0033     5        0
##       0.0001     7        0
##       0.0162     2        0
##       0.0009     2        0
##       0.0022     2        0
##       0.0010     2        0
##       0.0088     2        0
##       0.0017     2        0
##       0.0065     2        0
##       0.0244     1        0
##       0.0103     1        0
##       0.0108     1        0
##       0.0161     2        0
##       0.0015     2        0
##       0.0039     2        0
##       0.0010     2        0
##       0.0067     2        0
##       0.0014     2        0
##       0.0063     3        0
##       0.0000     3        0
##       0.0001     2        0
##       0.0016     2        0
##       0.0059     2        0
##       0.0006     3        0
##       0.0034     3        0
##       0.0009     3        0
##       0.0010     2        0
##       0.0185     1        0
##       0.0107     1        0
##       0.0070     1        0
##       0.0103     1        0
##       0.0044     1        0
##       0.0072     1        0
##       0.0067     4        0
##       0.0009     3        0
##       0.0020     2        0
##       0.0019     2        0
##       0.0007     4        0
##       0.0012     4        0
```

```
##       0.0007     4        0
##       0.0053     2        0
##       0.0053     2        0
##       0.0002     3        0
##       0.0000    17        0
##       0.0090     3        0
##       0.0138     2        0
##       0.0148     2        0
##       0.0052     2        0
##       0.0038     5        0
##       0.0027     7        0
##       0.0026     8        0
##       0.0208     2        0
##       0.0198     1        0
##       0.0073     2        0
##       0.0153     3        0
##       0.0099     1        0
##       0.0102     4        0
##       0.0042    17        0
##       0.0497     3        0
##       0.0515     2        0
##       0.0489     2        0
##       0.0598     2        0
##       0.0395     5        0
##       0.0273     7        0
##       0.0228     8        0
##       0.0626     2        0
##       0.0579     1        0
##       0.0426     2        0
##       0.0501     3        0
##       0.0442     1        0
##       0.0348     4        0
##       0.0025    17        0
##
## The design is not orthogonal
```

4.5.1 Questions

1. How does this analysis compare with that conducted in Section 3.3?

   The 42 df for Provenances[Countries] has been split into the differences between provenances for each country. Otherwise, the decompositions are the same.

5 Multiphase experiments in R

This class of experiments differs from those previously presented in that they often employ two or more randomizations or allocations, each to a different type of unit. As a result, there will be three or more sets of factors, or tiers, to deal with; further, when there are three sets of factors, three formula will need to be supplied to `designAnatomy`.

5.1 Athletic examples based on Brien et al. (2011)

Brien et al. (2011) give several designs for an athletic experiment that illustrate the basic principles to be employed in designing multiphase experiments. Here designs for two different multiphase scenarios are considered, both being based on a first-phase that is the testing phase and employs a split-unit design.

5.1.1 A standard single-phase athlete training experiment

First, a split-unit design is generated for an experiment in which the performance of an athlete when subject to nine different training conditions is tested. The nine training conditions are the combinations of three surfaces and three intensities of training. Also, assume that the prime interest is in surface differences, with intensities included to observe the surfaces over a range of intensities. The experiment is to involve 12 athletes, three per month for four consecutive months; each athlete undergoes three tests. The heart rate of the athlete is to be taken immediately upon completion of a test.

A split-plot design is to be employed for the experiment: the three intensities are randomized to the three athletes in each month and the three surfaces are randomized to the three tests that each athlete is to undergo. The factor-allocation diagram is shown in Figure 12. Generate a randomized layout for the experiment.

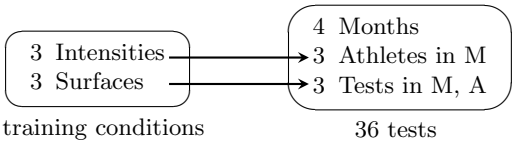
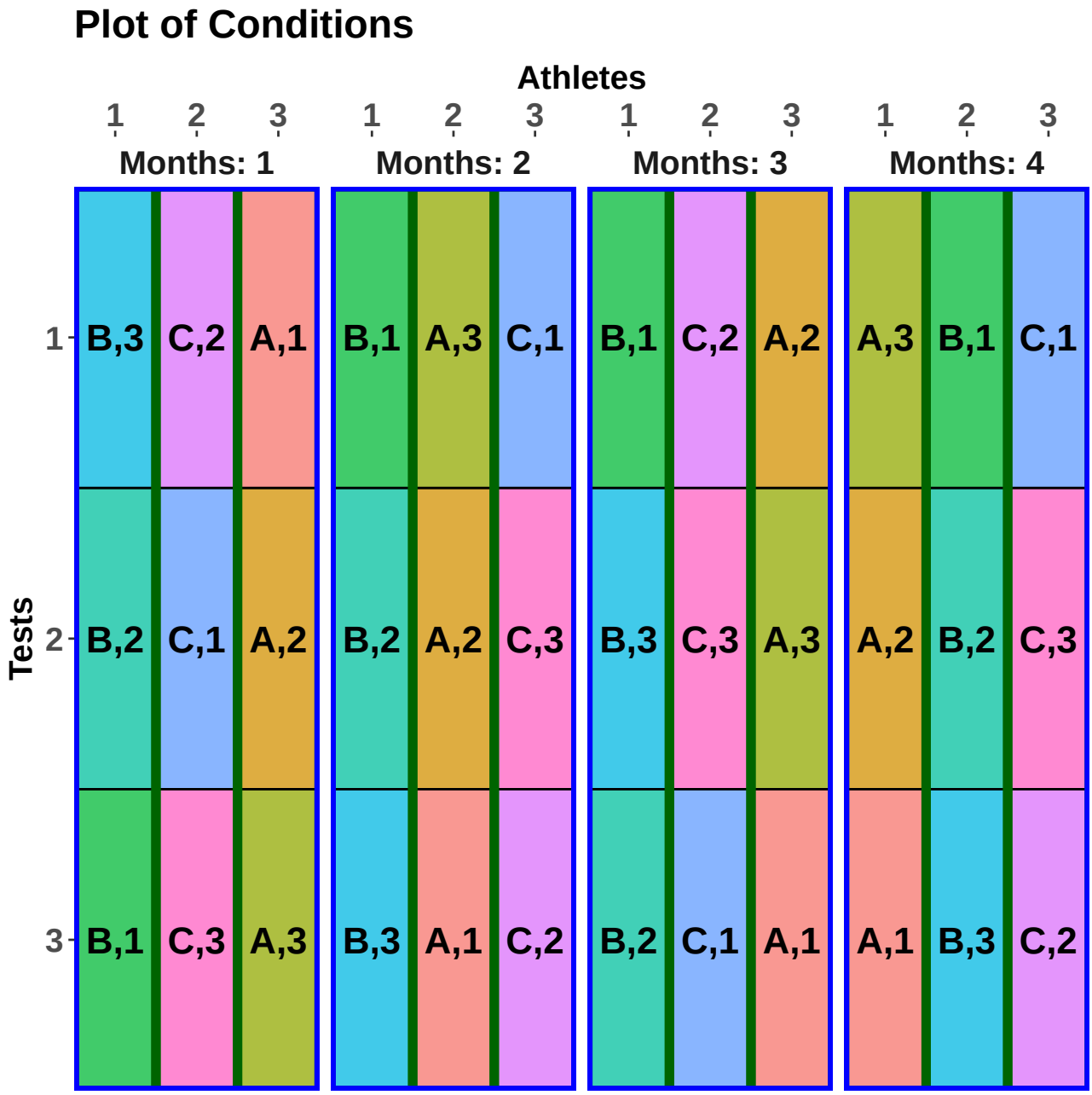


Figure 12: Factor-allocation diagram for the standard athlete training experiment: training conditions are randomized to tests; the two left-hand arrows indicate that the levels of Intensities and Surfaces are randomized to Athletes and Tests, respectively; M = Months; A = Athletes.

```
#'## Phase 1: Construct a systematic layout and generate a randomized layout for the first phase
split.sys <- cbind(fac.gen(list(Months = 4, Athletes = 3, Tests = 3)),
                   fac.gen(list(Intensities = LETTERS[1:3], Surfaces = 3),
                           times = 4))
split.lay <- designRandomize(allocated         = split.sys[c("Intensities", "Surfaces")],
                             recipient         = split.sys[c("Months", "Athletes", "Tests")],
                             nested.recipients = list(Athletes = "Months",
                                                      Tests = c("Months", "Athletes")),
                             seed              = 2598)
split.lay

##    Months Athletes Tests Intensities Surfaces
## 1       1        1     1           B        3
## 2       1        1     2           B        2
## 3       1        1     3           B        1
## 4       1        2     1           C        2
## 5       1        2     2           C        1
## 6       1        2     3           C        3
## 7       1        3     1           A        1
## 8       1        3     2           A        2
```

```
## 9       1        3     3           A        3
## 10      2        1     1           B        1
## 11      2        1     2           B        2
## 12      2        1     3           B        3
## 13      2        2     1           A        3
## 14      2        2     2           A        2
## 15      2        2     3           A        1
## 16      2        3     1           C        1
## 17      2        3     2           C        3
## 18      2        3     3           C        2
## 19      3        1     1           B        1
## 20      3        1     2           B        3
## 21      3        1     3           B        2
## 22      3        2     1           C        2
## 23      3        2     2           C        3
## 24      3        2     3           C        1
## 25      3        3     1           A        2
## 26      3        3     2           A        3
## 27      3        3     3           A        1
## 28      4        1     1           A        3
## 29      4        1     2           A        2
## 30      4        1     3           A        1
## 31      4        2     1           B        1
## 32      4        2     2           B        2
## 33      4        2     3           B        3
## 34      4        3     1           C        1
## 35      4        3     2           C        3
## 36      4        3     3           C        2

#'## Get anatomy to check properties of the design
split.canon <- designAnatomy(formulae = list(tests = ~ Months/Athletes/Tests,
                                             cond  = ~ Intensities*Surfaces),
                             data     = split.lay)
summary(split.canon, which.criteria="none")

##
##
## Summary table of the decomposition for tests & cond
##
##  Source.tests           df1 Source.cond          df2
##  Months                   3
##  Athletes[Months]         8 Intensities            2
##                             Residual               6
##  Tests[Months:Athletes]  24 Surfaces               2
##                             Intensities#Surfaces   4
##                             Residual              18

#'## Plot the design
#+ "SplitDes_v2"
split.lay <- within(split.lay,
                    Conditions <- fac.combine(list(Intensities, Surfaces),
                                              combine.levels = TRUE))
plt <- designGGPlot(split.lay, labels = "Conditions",
                    row.factors = "Tests", column.factors = c("Months", "Athletes"),
```

```
                    cellalpha = 0.75, size = 6,
                    blockdefinition = rbind(c(3,1)), blocklinecolour = "darkgreen",
                    printPlot = FALSE)
designBlocksGGPlot(plt, nrows = 3, ncolumns = 3, blockdefinition = rbind(c(3,3)))
```



Question

1. Why was a split-plot design chosen for this experiment?

   Because it is likely that variation between tests within an athlete will be smaller than variation between athletes within a month. Hence, because the prime interest is in Surfaces, they are assigned to tests within an athlete and will have better precision than Intensities, which have been assigned to the more variable

athletes within a month.

5.1.2 A simple two-phase athlete training experiment

Suppose that, in addition to heart rate taken immediately upon completion of a test, the free haemoglobin is to be measured using blood specimens taken from the athletes after each test and transported to the laboratory for analysis. That is, a second laboratory phase is required to obtain the new response. In this phase, because the specimens become available monthly, the batch of specimens for one month are to be processed, in a random order, before those for the next month are available. The factor-allocation diagram for this experiment is in Figure 13, the dashed line indicating that Months are systematically allocated to Batches. The randomizations in this diagram are composed (Brien and Bailey, 2006) and is one of the two types of randomizations in a chain (Bailey and Brien, 2016). This means that the second-phase randomization only need to consider how the tests factors are to be assigned to locations; training conditions can be ignored in determining the second-phase design.

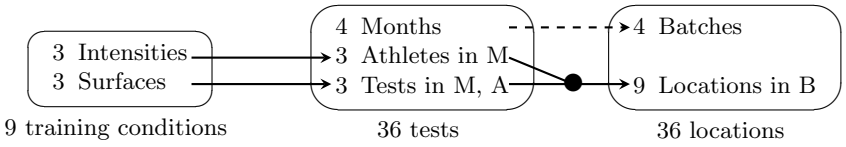


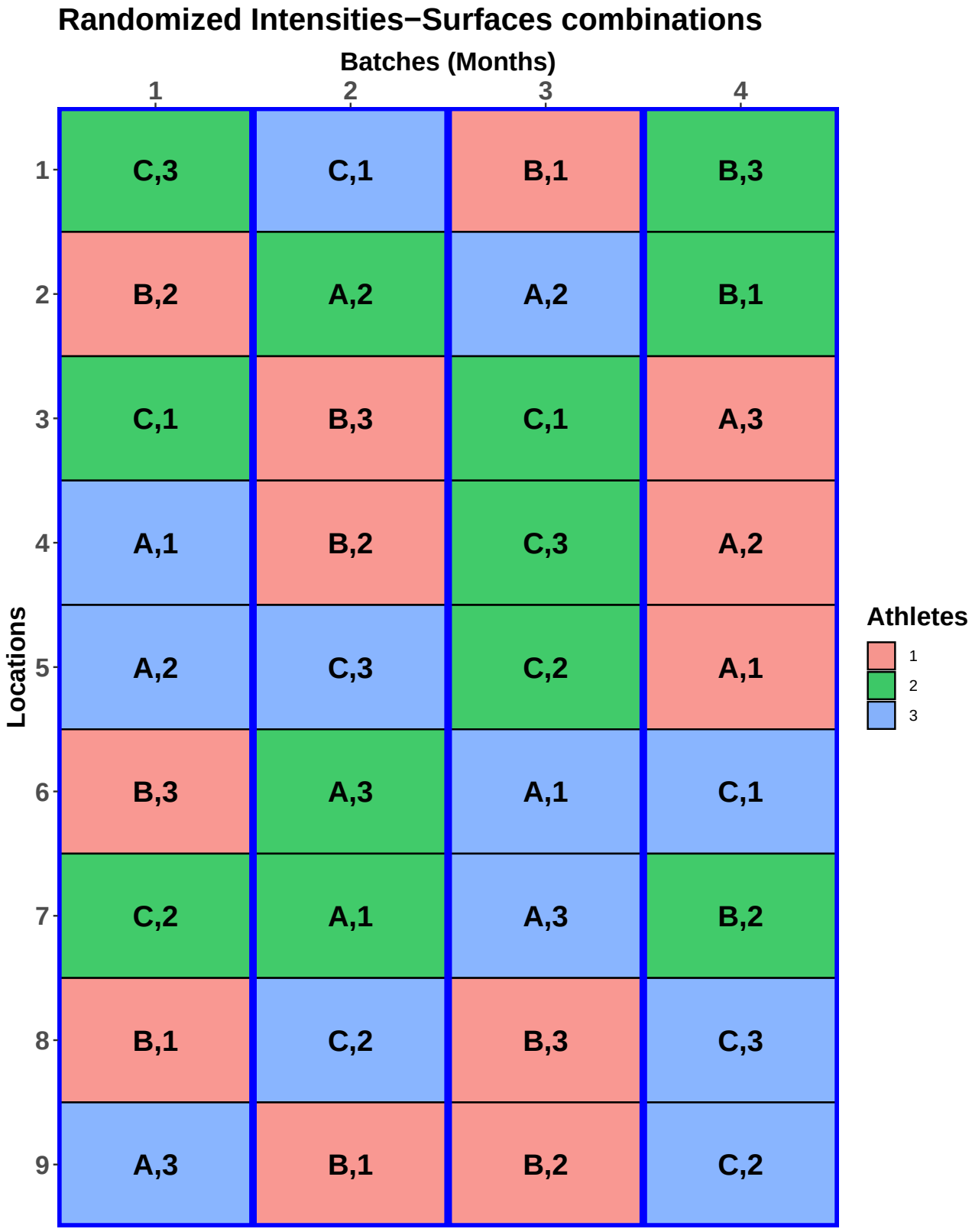
Figure 13: Factor-allocation diagram for the two-phase athlete training experiment: training conditions are randomized to tests and tests are allocated to locations; the two left-hand arrows indicate that the levels of Intensities and Surfaces are randomized to Athletes and Tests, respectively; the dashed arrow indicates that Months are systematically allocated to Batches; the '●' indicates that the combinations of the levels of Athletes and Tests are randomized to the Locations; M = Months; A = Athletes; B = Batches.

Using the following R code, obtain a layout for the second phase and check the properties of the layout. In doing this, the first-phase layout is randomized. However, because Months is not randomized to Batches, the argument `except` in `designRandomize` is used to effect the systematic allocation.

```
#'# Generate a layout for a simple two-phase athlete training experiment
#'
#'## Phase 1 - the split-plot design that has already been generated.
#'## Phase 2 - randomize tests (and training conditions) to locations,
#'##           but Months assigned systematically to Batches
#'##           so except Batches from the randomization
eg1.lay <- designRandomize(allocated         = split.lay,
                           recipient         = list(Batches = 4, Locations = 9),
                           nested.recipients = list(Locations = "Batches"),
                           except            = "Batches",
                           seed              = 71230)
eg1.lay

##    Batches Locations Months Athletes Tests Intensities Surfaces Conditions
## 1        1         1      1        2     3           C        3        C,3
## 2        1         2      1        1     2           B        2        B,2
## 3        1         3      1        2     2           C        1        C,1
## 4        1         4      1        3     1           A        1        A,1
## 5        1         5      1        3     2           A        2        A,2
## 6        1         6      1        1     1           B        3        B,3
## 7        1         7      1        2     1           C        2        C,2
## 8        1         8      1        1     3           B        1        B,1
## 9        1         9      1        3     3           A        3        A,3
## 10       2         1      2        3     1           C        1        C,1
## 11       2         2      2        2     2           A        2        A,2
## 12       2         3      2        1     3           B        3        B,3
## 13       2         4      2        1     2           B        2        B,2
```

```
## 14       2         5      2         3     2          C        3        C,3
## 15       2         6      2         2     1          A        3        A,3
## 16       2         7      2         2     3          A        1        A,1
## 17       2         8      2         3     3          C        2        C,2
## 18       2         9      2         1     1          B        1        B,1
## 19       3         1      3         1     1          B        1        B,1
## 20       3         2      3         3     1          A        2        A,2
## 21       3         3      3         2     3          C        1        C,1
## 22       3         4      3         2     2          C        3        C,3
## 23       3         5      3         2     1          C        2        C,2
## 24       3         6      3         3     3          A        1        A,1
## 25       3         7      3         3     2          A        3        A,3
## 26       3         8      3         1     2          B        3        B,3
## 27       3         9      3         1     3          B        2        B,2
## 28       4         1      4         2     3          B        3        B,3
## 29       4         2      4         2     1          B        1        B,1
## 30       4         3      4         1     1          A        3        A,3
## 31       4         4      4         1     2          A        2        A,2
## 32       4         5      4         1     3          A        1        A,1
## 33       4         6      4         3     1          C        1        C,1
## 34       4         7      4         2     2          B        2        B,2
## 35       4         8      4         3     2          C        3        C,3
## 36       4         9      4         3     3          C        2        C,2
```

```
#'## Plot the layout
#+ Athlete_eg1lay
eg1.lay$Conditions <- with(eg1.lay, fac.combine(list(Intensities, Surfaces),
                                                combine=TRUE, sep=","))
designGGPlot(eg1.lay, labels = "Conditions",
             row.factors = "Locations", column.factors = "Batches",
             cellfillcolour.column = "Athletes", cellalpha = 0.75,  size = 6,
             title = "Randomized Intensities-Surfaces combinations",
             blockdefinition = rbind(c(9,1)),
             ggplotFuncs = list(xlab("Batches (Months)"),
                                theme(legend.position = "right")))
```



Check the properties of the design.

```
#'## Check properties of the design
eg1.canon <- designAnatomy(formulae = list(locs  = ~ Batches/Locations,
                                           tests = ~ Months/Athletes/Tests,
                                           cond  = ~ Intensities*Surfaces),
                           data     = eg1.lay)
summary(eg1.canon, which.criteria="none")

##
##
## Summary table of the decomposition for locs, tests & cond
##
##  Source.locs        df1 Source.tests           df2 Source.cond          df3
##  Batches              3 Months                   3
##  Locations[Batches]  32 Athletes[Months]         8 Intensities            2
##                                                   Residual               6
##                         Tests[Months:Athletes]  24 Surfaces               2
##                                                   Intensities#Surfaces   4
##                                                   Residual              18
```

Questions

1. What would be the allocation-based mixed model for this experiment, an allocation-based mixed model having the same terms as the randomization-based mixed model that would apply if all the allocations had been made by randomizing. Do you anticipate any problem in fitting it?

   The allocation-based mixed model is formed by treating all training-conditions factors as fixed and the remaining factors as random. Hence, the symbolic mixed model is Intensities + Surfaces + Intensities^Surfaces | Months + Months^Athletes + Months^Athletes^Tests + Batches + Batches^Locations. The problem in fitting it would be that Months and Batches are confounded so that the variance model is singular.

2. Compare the units for the two phases in this experiment?

   A unit in the first phase is a test conducted on an athlete in a particular month; in the second phase, a unit is a location of a test within a batch. That is, the unit in the first phase is an athlete's test and in the second phase is a blood specimen in a lab location.

3. What are the outcomes for the two phases for this experiment?

   The outcome for the first phase is the heart rate for a test and a blood specimen from the test; the outcome for the second phase, is the free haemglobin measured at a location.

5.1.3 Allowing for lab processing order in the athletic training example

Brien (2017) discusses a design, and its properties, that differs in the second phase from that described in Section 5.1.2: it assumes that lab processing order within a batch is important and so the second phase now requires a row-column design. However, one cannot consider a design for just Months, Athletes and Tests and ignore Intensities and Surfaces, as was done in the previous design. Indeed prime consideration needs to be given to Intensities and Surfaces. That is, a suitable cross-phase design for allocating Intensities and Surfaces to Batches and Locations is needed. However, the second-phase design that allocates Months, Athletes and Tests to Batches and Locations has to be considered in that it must account for the split-unit nature of the first-phase design.

For the second-phase design, the Months are associated with Batches. Then each triple of consecutive locations in a batch are associated with a single athlete, one of those for the month associated with the batch. This leaves tests to be assigned to locations within triples. Thus, the cross-phase design will need to allocate efficiently an intensity to a location triple and surface to the locations within a triple.

The cross-phase design is a balanced factorial design (Hinkelmann and Kempthorne, 2005, Section 12.5) and can be constructed using two extended Latin squares (ELS) as follows:

1. a 3×4 ELS, formed from a 3×3 Latin square by repeating one of its columns, will be used to allocate Intensities to the 3 Locations triples $\times$ 4 Months.

2. A 3×4 ELS will be used to allocate Surfaces to the 3 Locations $\times$ 4 Months within a triple; the same ELS is used for the three triples.

3. To ensure no repeat Intensities-Surfaces combinations for a Location, the two Batches to which the repeated columns of the ELS for Intensities are assigned must be different from the two Batches to which repeated columns of the ELS for Surfaces are assigned.

The factor-allocation diagram, for this design, is in Figure 14. In this diagram, the training conditions and tests panels are surrounded by a dashed rectangle and lines go from the training conditions sources to the lines from the test sources. This indicates that the result of the allocation in the first phase needs to be explicitly taken into account in the second-phase allocation. The randomizations involved have been called a randomized-inclusive randomizations (Brien and Bailey, 2006) and are one of the two types of randomizations in a chain (Bailey and Brien, 2016). Because Batches and Locations are crossed, the second phase randomization is achieved by independently permuting the Batches and Locations. A design with the same properties had been previously constructed by Rosemary Bailey (pers. comm.).

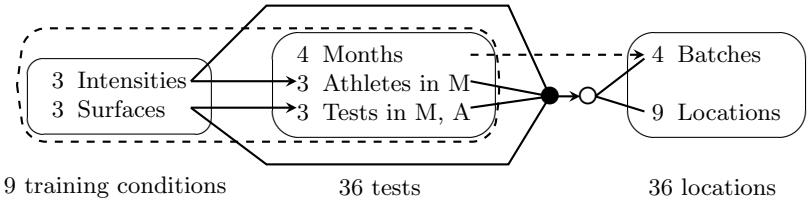
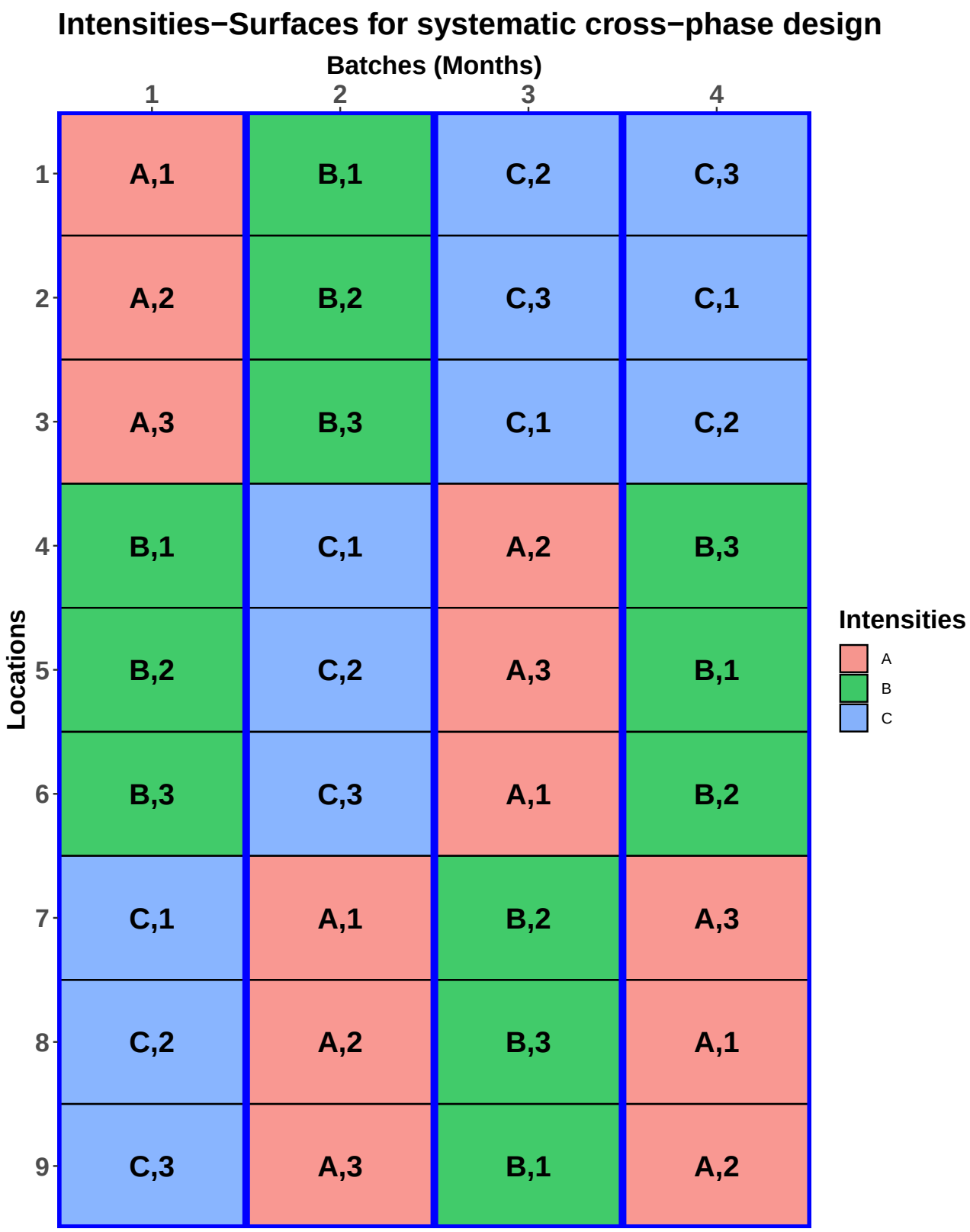


Figure 14: Factor-allocation diagram for the two-phase athlete training experiment with a row-column design for the second phase: training conditions are randomized to tests, then training conditions and tests are randomized to locations; the '●' indicates that the observed combinations of the levels of Intensities, Surfaces, Athletes and Tests are randomized to locations; the '○' indicates that a nonorthogonal design was used in this randomization to the combinations of the levels of Batches and Locations; the dashed arrow indicates that Months were systematically allocated to Batches; the dashed oval indicates that all factors from the first phase form a pseudotier and all are actively involved in determining the allocation to locations; M = Months and A = Athletes.

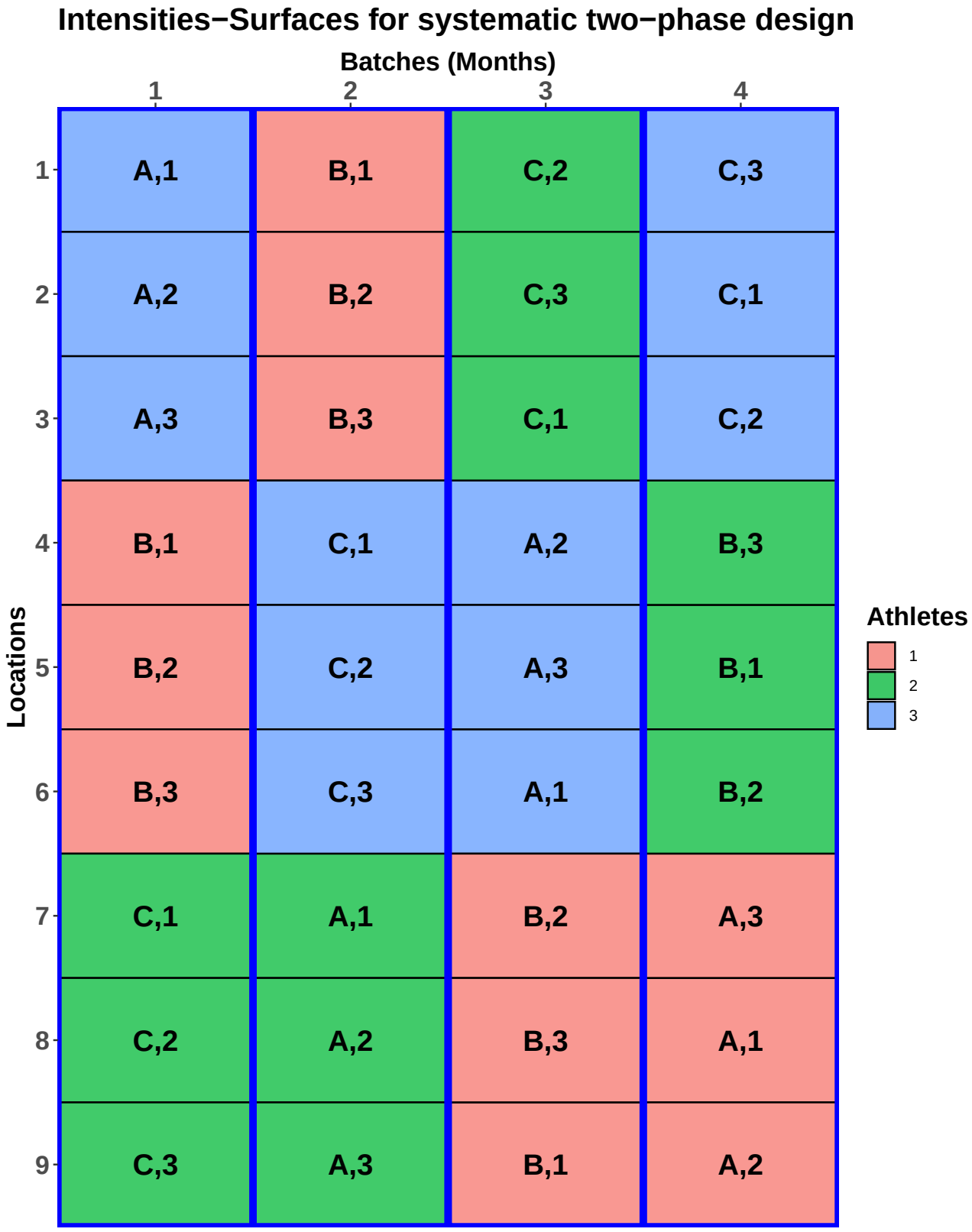
Use the following R code to obtain a layout for the new second phase design.

```
#'## Generate a systematic cross-phase design for Intensities and Surfaces
#' It is based on (i) an extended Latin square (ELS) for allocating Intensities to
#' Locations triples x Batches and (ii) the same ELS for each triple, the ELSD being used to
#' allocate Surfaces to the three Locations within each triple by four Batches.
#' The Batches to which the repeated columns of the ELSD for Intensities are assigned must be
#' different from the Batches to which repeated columns of the ELSD for Surfaces are assigned.
#+ Athlete_eg2sys_v3
eg2.phx.sys <- cbind(fac.gen(list(Batches = 4, Locations = 9)),
                     data.frame(Intensities = factor(rep(c(designLatinSqrSys(3), c(3,2,1)),
                                                         each = 3), labels = LETTERS[1:3]),
                                Surfaces = factor(c(rep(1:3, times = 3),
                                                    rep(1:3, times = 3),
                                                    rep(c(2,3,1), times = 3),
                                                    rep(c(3,1,2), times = 3)))))
eg2.phx.sys$Conditions <- with(eg2.phx.sys, fac.combine(list(Intensities, Surfaces),
                                                        combine.levels = TRUE))
designGGPlot(eg2.phx.sys, labels = "Conditions",
             row.factors = "Locations", column.factors = "Batches",
             cellfillcolour.column = "Intensities", cellalpha = 0.75, size = 6,
             title = "Intensities-Surfaces for systematic cross-phase design",
             blockdefinition = rbind(c(9,1)),
```

```
        ggplotFuncs = list(xlab("Batches (Months)"),
                           theme(legend.position = "right")))
```

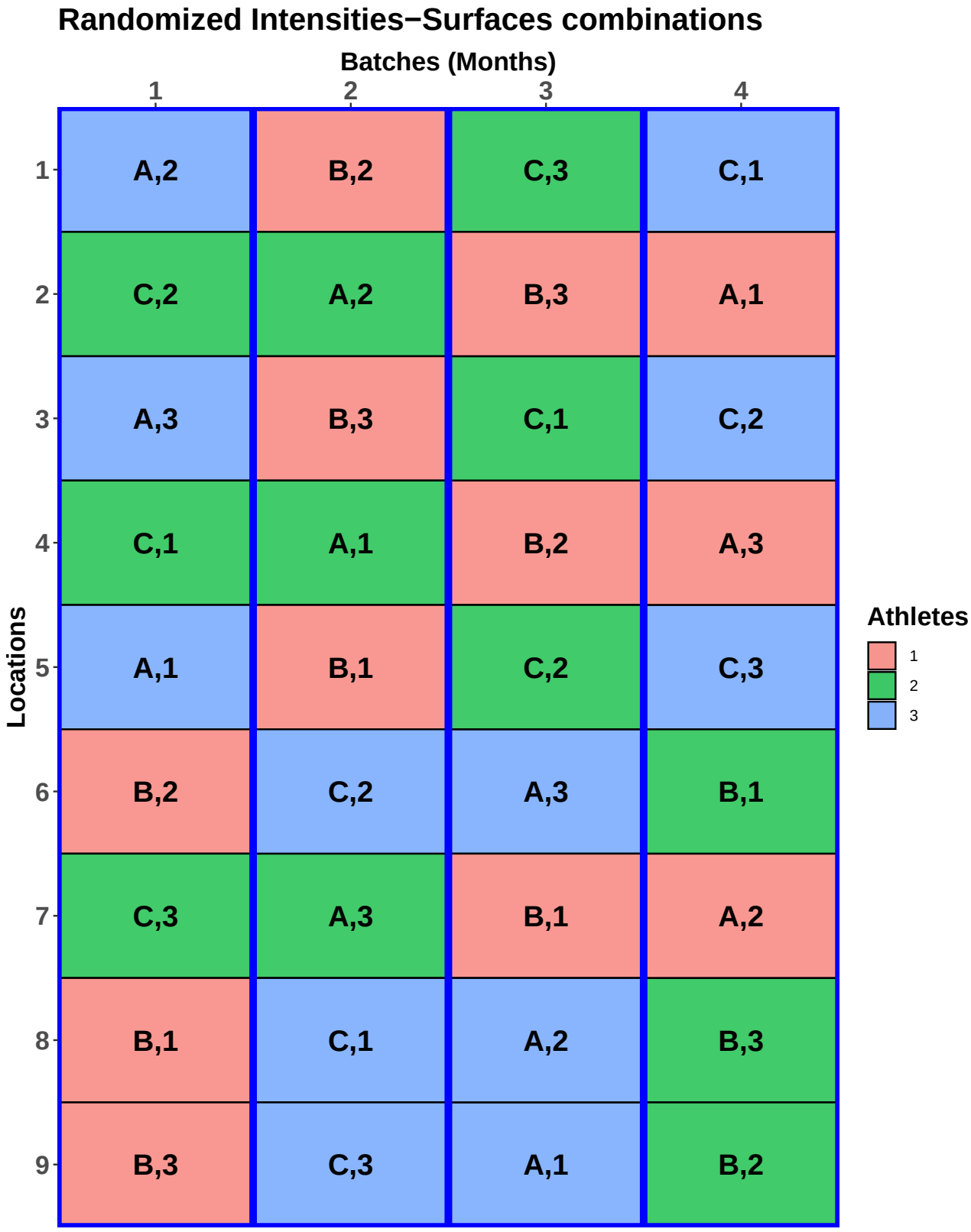


```
#'## Second phase design
#'## Generate a systematic two-phase design by bringing in first-phase recipient factors
eg2.phx.sys$Months <- eg2.phx.sys$Batches
eg2.sys <- merge(split.lay, eg2.phx.sys) #merge on common factors Months, Intensities & Surfaces
designGGPlot(eg2.sys, labels = "Conditions",
             row.factors = "Locations", column.factors = "Batches",
             cellfillcolour.column = "Athletes", cellalpha = 0.75,  size = 6,
             title = "Intensities-Surfaces for systematic two-phase design",
             blockdefinition = rbind(c(9,1)),
             ggplotFuncs = list(xlab("Batches (Months)"),
                                theme(legend.position = "right")))
```




```
#'## Allocate the second phase
eg2.lay <- designRandomize(allocated = eg2.sys[c("Months", "Athletes", "Tests",
```

```
                                                "Intensities", "Surfaces")],
                          recipient = eg2.sys[c("Batches", "Locations")],
                          except    = "Batches",
                          seed      = 243526)
head(eg2.lay)

##   Batches Locations Months Athletes Tests Intensities Surfaces
## 1       1         1      1        3     2           A        2
## 2       1         2      1        2     1           C        2
## 3       1         3      1        3     3           A        3
## 4       1         4      1        2     2           C        1
## 5       1         5      1        3     1           A        1
## 6       1         6      1        1     2           B        2

#'## Plot the layout
#+ Athlete_eg2lay_v3
eg2.lay$Conditions <- with(eg2.lay, fac.combine(list(Intensities, Surfaces),
                                                combine=TRUE, sep=","))
designGGPlot(eg2.lay, labels = "Conditions",
             row.factors = "Locations", column.factors = "Batches",
             cellfillcolour.column = "Athletes", cellalpha = 0.75,  size = 6,
             title = "Randomized Intensities-Surfaces combinations",
             blockdefinition = rbind(c(9,1)),
             ggplotFuncs = list(xlab("Batches (Months)"),
                                theme(legend.position = "right")))
```



Check the properties of the design.

```
#'## Check properties of the design
eg2.canon <- designAnatomy(formulae = list(locs  = ~ Batches*Locations,
                                           tests = ~ Months/Athletes/Tests,
                                           cond  = ~ Intensities*Surfaces),
                           data     = eg2.lay)
summary(eg2.canon, which.criteria =c("aefficiency", "order"))

##
##
## Summary table of the decomposition for locs, tests & cond (based on adjusted quantities)
##
##  Source.locs       df1 Source.tests           df2 Source.cond          df3 aefficiency order
##  Batches             3 Months                   3                            1.0000     1
##  Locations           8 Athletes[Months]         2 Intensities              2      0.0625     1
##                        Tests[Months:Athletes]   6 Surfaces                 2      0.0625     1
##                                                   Intensities#Surfaces     4      0.2500     1
##  Batches#Locations  24 Athletes[Months]         6 Intensities              2      0.9375     1
##                                                   Residual                 4      1.0000     1
##                        Tests[Months:Athletes]  18 Surfaces                 2      0.9375     1
##                                                   Intensities#Surfaces     4      0.7500     1
##                                                   Residual                12      1.0000     1
##
## The design is not orthogonal
```

It is clear that Athletes[Months] and Tests[Months:Athletes] are not orthogonal to Locations and Batches#Locations, because the former sources are confounded with both of the latter sources. To examine the nature of the nonorthogonality, the skeleton anova for just the tests and locations tiers is obtained.

```
#"### Examine the nonorthogonality between locations and tests
eg2.locstests.canon <- designAnatomy(formulae = list(locs  = ~ Batches*Locations,
                                                     tests = ~ Months/Athletes/Tests),
                                     data     = eg2.lay)
summary(eg2.locstests.canon, which.criteria =c("aefficiency", "order"))

##
##
## Summary table of the decomposition for locs & tests
##
##  Source.locs       df1 Source.tests           df2 aefficiency order
##  Batches             3 Months                   3      1.0000     1
##  Locations           8 Athletes[Months]         2      1.0000     1
##                        Tests[Months:Athletes]   6      1.0000     1
##  Batches#Locations  24 Athletes[Months]         6      1.0000     1
##                        Tests[Months:Athletes]  18      1.0000     1
```

Questions

1. What do you conclude about the confounding of Athletes[Months] and Tests[Months:Athletes] with Locations?

   Since all efficiency factors are one, it is concluded that the 8 degrees of freedom for Athletes[Months] has been split into two orthogonal parts, one with 2 degrees of freedom which is confounded with Batches and the other with 6 degrees of freedom which is confounded with Batches:Locations. The source Tests[Months:Athletes] has been similarly partitioned.

2. Are the designs proposed for this experiment first-order balanced?

   The design is first-order balanced, because the order of the efficiency factors is one for all confounded sources.

3. What has been the cost of allowing for order of processing in the lab? Is the cost acceptable? Why?

   The cost has been that some information about Athletes[Months], along with Intensities, and some information about Tests[Months:Athletes], along with Surfaces and Intensities#Surfaces, has been confounded with Locations. The cost is acceptable, because the amount of information lost on the main effects is only 6.25% and on the interaction is 25%. The latter will be recovered in a REML-based mixed model analysis. However, the Residual degrees of freedom for Athletes[Months] has been reduced from 6 to 4 and for Tests[Months:Athletes] from 18 to 14. While the latter is unlikely to be seriously deleterious, the former is of concern.

5.2 McIntyre's (1955) two-phase example

McIntyre (1955) reports an investigation of the effect of four light intensities on the synthesis of tobacco mosaic virus in leaves of tobacco *Nicotiana tabacum* var. Hickory Pryor. It is a two-phase experiment: the first phase is a treatment phase, in which the four light treatments are randomized to the tobacco leaves, and the second phase is an assay phase, in which the tobacco leaves are randomized to the half-leaves of assay plants.

In the first phase, four successive leaves at defined positions on the stem were taken from each of eight plants of comparable age and vigour that had been inoculated with the virus. Arbitrarily grouping the plants into two sets of four, the four treatments were applied to the leaves, which had been separated from the plants and were sustained by flotation on distilled water, in a Latin square design for each set with tobacco plants as columns and leaf positions as rows; see Figure 16.

In the second phase, virus content of each tobacco leaf was assayed by expressing sap and inoculating half leaves of the assay plants, *Datura stramonium*, on which countable lesions would appear. Lots of eight sap samples were formed from pairs of tobacco plants, the pairs being comprised of a plant from each set in the treatment phase. The eight samples from a lot were assigned to four assay plants using one of four 4×4 Graeco-Latin square designs, with the leaves from a single tobacco plant assigned using one of the alphabets and the second tobacco plant using the other (see Figure 17). Actually, this design is a semi-Latin square (Bailey, 1992).

The factor-allocation diagram for the experiment is in Figure 15. Unfortunately, the randomization for this experiment was not described by McIntyre (1955). Because there are multiple squares in both phases, there are several possible randomizations depending on the effects anticipated as possible in the experiment. As shown by the nesting relations in the factor-allocation diagram, I have assumed that randomization to NicPlant was within Sets and to Posn was across Sets. Similarly, I have assumed that randomization to DatPlant was within Lot and to AssPosn across Lot. In the factor-allocation diagram, N_1 is a factor for the pairs of tobacco plants formed by taking a plant from each set in the first phase.

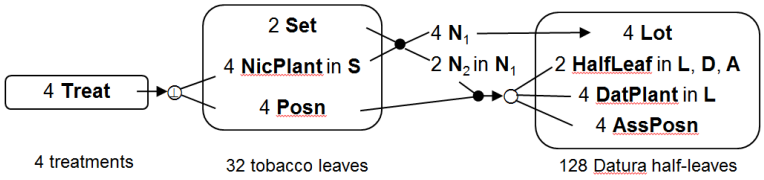


Figure 15: Factor-allocation diagram for McIntyre's (1955) two-phase experiment: treatments are randomized to tobacco leaves and tobacco leaves are randomized to Datura half-leaves; the arrow to the '⊥', the '⊥' and the two lines from the '⊥' indicate that Treat is randomized to the combinations of NicPlant and Posn using an orthogonal design; N_1 is a pseudofactor indexing the pairs of tobacco plants formed by taking a plant from each set in the first phase and N_2 is a pseudofactor indexing the tobacco plants within the pairs formed by taking a plant from each set in the first phase; N_1 is randomized to Lot in the second phase; the combinations of N_2 and Posn is randomized to the combinations of HalfLeaf, DatPlant and AssPosn using a nonorthogonal design, the latter indicated by the 'O'; S = Set; L = Lot; D = DatPlant; A = AssPosn.

Figure 16: Layout for the first phase of McIntyre's (1955) experiment†

Nicotiana Plants

Leaf Position	1	2	3	4	Leaf Position	1	2	3	4
1	a	b	c	d		a	b	c	d
	1	5	9	13		17	21	25	29
2	b	a	d	c		c	d	a	b
	2	6	10	14		18	22	26	30
3	c	d	a	b		d	c	b	a
	3	7	11	15		19	23	27	31
4	d	c	b	a		b	a	d	c
	4	8	12	16		20	24	28	32

†The letter in each cell refers to the light intensity to be applied to the unit and the number to the unit.

Figure 17: Layout for the second phase of McIntyre's (1955) experiment†

Datura Plants

Assay Leaf Position	1	2	3	4	Assay Leaf Position	5	6	7	8
1	1	2	3	4		5	6	7	8
	17	20	18	19		23	22	24	21
2	2	1	4	3		8	7	6	5
	18	19	17	20		22	23	21	24
3	3	4	1	2		7	8	5	6
	19	18	20	17		21	24	22	23
4	4	3	2	1		6	5	8	7
	20	17	19	18		24	21	23	22

Datura Plants

Assay Leaf Position	9	10	11	12	Assay Leaf Position	13	14	15	16
1	9	10	11	12		13	14	15	16
	28	25	27	26		30	31	29	32
2	10	9	12	11		16	15	14	13
	27	26	28	25		31	30	32	29
3	11	12	9	10		15	16	13	14
	26	27	25	28		32	29	31	30
4	12	11	10	9		14	13	16	15
	25	28	26	27		29	32	30	31

†The numbers in the cell refer to the units from the first phase (tobacco leaves) to be assigned to the two half-leaves of the assay plant; they are in standard order for Set, then NicPlant followed by Position.

5.2.1 Check the properties of the randomized layout

Load the data and use `designAnatomy` to check the properties of the design.

```
#'## Load data
data("McIntyreTMV.dat")
#'## Check properties of the design
TMV.canon <- designAnatomy(formulae = list(assay = ~ ((Lot/DatPlant)*AssPosn)/HalfLeaf,
                                           test  = ~ (Set/NicPlant)*Posn,
                                           trt   = ~ Treat),
                           data     = McIntyreTMV.dat)
summary(TMV.canon, which.criteria=c("aeff", "ord"))

##
##
## Summary table of the decomposition for assay, test & trt (based on adjusted quantities)
##
##  Source.assay                    df1 Source.test        df2 Source.trt df3 aefficiency order
##  Lot                               3 NicPlant[Set]        3                     1.0000     1
##  DatPlant[Lot]                    12
##  AssPosn                           3
##  Lot#AssPosn                       9
##  DatPlant#AssPosn[Lot]            36 Posn                   3                     0.5000     1
##                                      Set#Posn               3                     0.5000     1
##                                      NicPlant#Posn[Set]    18 Treat         3      0.5000     1
##                                                               Residual     15      0.5000     1
##                                      Residual              12
##  HalfLeaf[Lot:DatPlant:AssPosn]   64 Set                    1                     1.0000     1
##                                      NicPlant[Set]          3                     1.0000     1
##                                      Posn                   3                     0.5000     1
##                                      Set#Posn               3                     0.5000     1
##                                      NicPlant#Posn[Set]    18 Treat         3      0.5000     1
##                                                               Residual     15      0.5000     1
##                                      Residual              36
##
## The design is not orthogonal
```

5.2.2 Questions

1. Is the variance matrix for this experiment based on two sets of terms that are orthogonal?

   The variance matrix for this experiment is based on the factors in the tobacco leaves and Datura half-leaves tiers. The terms derived from the factors in these two tiers are not orthogonal. In particular, Set#Posn and NicPlant#Posn[Set] are partially confounded with both DatPlant#AssPosn[Lot] and HalfLeaf[Lot:DatPlant:AssPosn].

2. What are the advantages and disadvantages of a mixed-model analysis of the data from this experiment, as opposed to an anova?

   The advantage of a mixed-model analysis is that combined estimates will be provided for Set#Posn, NicPlant#Posn[Set], and Treat. The disadvantages are (i) that not all random terms are well-estimated, some having small degrees of freedom, and cause problems in fitting the model, and (ii) the Wald F-statistics are only approximately distributed as F-distributions. On the other hand, an anova is not applicable because of the nonorthogonality between the sets of terms making up the variance matrix; at least some F-ratios will not be independently distributed.

5.3 A Plant Accelerator experiment with a split-unit design

This experiment involves the investigation of 75 wheat lines, of which 73 are Nested Association Mapping (NAM) wheat lines and the other two are two check lines, Scout and Gladius. It was conducted in 2014 in the Plant Accelerator, a facility in Adelaide with 4 Smarthouses. A Smarthouse is a large greenhouse with two areas within it: (i) a Table area at the southern end and (ii) a Conveyor area at the northern end — see Figure 18. The conveyor system has the capability of automatically moving and imaging around 500 pots per day. There are air conditioners placed down the western side of the Smarthouse, which creates a trend from west to east. Further, there is a north-south trend due to changes in light intensity (Brien et al., 2013).

The experiment involves two phases: the table and conveyor phases. The table phase is the establishment phase in which plants are germinated in pots on the tables where they undergo an early growth stage. In the conveyor phase, having placed the pots in carts on the conveyor system, the plants are automatically imaged and watered daily, being moved to a processing station by the conveyor system for this.

This experiment has a single plant per pot and these will be arranged in a 24×22 grid in both phases: 24 columns $\times$ 22 locations in the table phase and 24 lanes $\times$ 22 (2–23) positions, as shown in Figure 18. However, the 24 columns in the table phase run east-west and the 24 lanes in the conveyor phase run north-south. Because there are systematic trends in both phases to be accounted for in the analysis, the same layout will be used in both phases, but the table layout will be rotated clockwise through 90°. That is, Locations 1–22 will be in Positions 2–23, respectively, and the Column will be placed in the Lane with the same number.

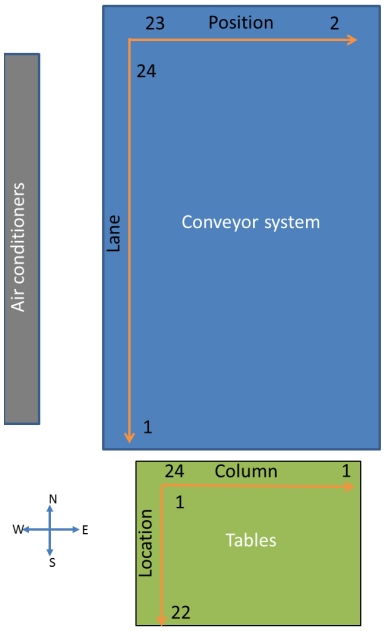


Figure 18: Schematic of Smarthouse for the Plant Accelerator experiment

The design employed for the experiment is a split-unit design in which two consecutive pots/carts form a main unit. The main-unit design uses a blocked design with rows and columns generated with `DiGGer` (Coombes, 2009). It assigns Lines to main units, the Lines being unequally replicated; Scout and Gladius each occur on 12 main units (24 carts), 21 randomly-selected NAM lines each occur on 4 main units (8 carts) and the remaining 52 NAM lines each occur on 3 main units (6 carts). The subunit design merely randomizes Salt (0 mM NaCl, 100 mM NaCl) to the two carts in each main unit.

In the main-unit design, the blocks are, in the table phase, 6 Groups of 4 Columns and, in the conveyor phase, 6 Zones of 4 Rows (lanes). However, while the generated design is based on crossed rows and columns, it is known from past experience that, while there are differences between Zones, there are not differences between Rows within Zones (Brien et al., 2013) and none are anticipated between Columns within Groups on the tables. The columns of the main-unit design are indexed by 11 Pairs in the table phase and 11 MainPosns in the conveyor phase. The design generated with `DiGGer` (Coombes, 2009) will be rerandomized so that the Lines are

randomized to 4 Columns within each Groups-Pairs combination and the 11 sets of Lines assigned by `DiGGer` to the 11 Pairs will be rerandomized to Pairs. The factor-allocation diagram is shown in Figure 19.

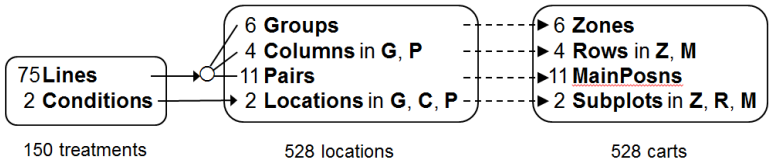


Figure 19: Factor-allocation diagram for the Plant Accelerator experiment: treatments are randomized to locations and locations are allocated to cars; the arrow to the 'O', the 'O' and the three lines from the 'O' indicate that Lines is randomized to the combinations of Groups, Columns and Pairs using a nonorthogonal design; the arrow from Conditions to Locations indicates that Conditions were randomized to Locations; the dashed arrows between the two panels on the right hand side indicate that the factors indexing locations were systematically assigned to those indexing the carts; G = Groups; P = Pairs; C = Columns; Z = Zones; M = MainPosns; R = Rows.

5.3.1 Produce the layout

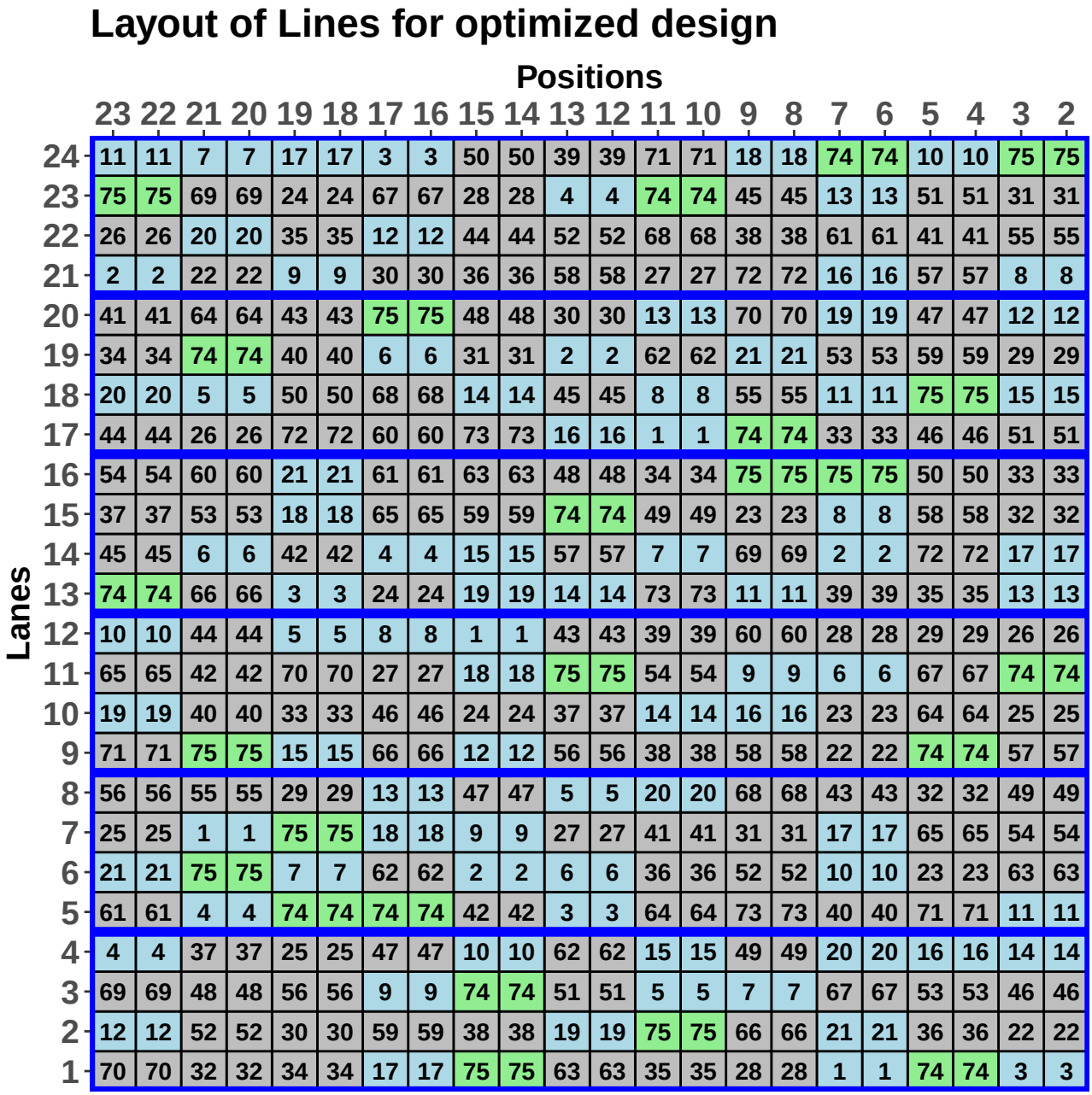
Use the following instructions to load the main-unit design produced with `DiGGer` and check its properties using `designAnatomy`.

```
#'## Load the main-unit design - it has Lines in row-column order
data("Exp249.munit.des")
Exp249.munit.des$Blocks <- factor(rep(1:6, each = 44))

#'## Check its properties
Exp249.munit.canon <- designAnatomy(formulae = list(cart  = ~ (Blocks/Rows)*Cols,
                                                    treat = ~ Lines),
                                    data     = Exp249.munit.des)
summary(Exp249.munit.canon)

##
##
## Summary table of the decomposition for cart & treat (based on adjusted quantities)
##
##  Source.cart       df1 Source.treat df2 aefficiency eefficiency order
##  Blocks              5 Lines          5      0.1498      0.1422     5
##  Rows[Blocks]       18 Lines         18      0.2685      0.1813    18
##  Cols               10 Lines         10      0.2102      0.1769    10
##  Blocks#Cols        50 Lines         50      0.1154      0.0142    50
##  Rows#Cols[Blocks] 180 Lines         74      0.5816      0.2088    74
##                        Residual     106
##
## The design is not orthogonal
```

Expand main-unit design to produce the split-unit design, including a three-level factor Checks that compares Scout, Gladius and the mean of the NAM lines. Perhaps, produce a plot of the allocation of the Lines.

```
#'## Expand design to rerandomize lines and to assign salt treatments to locations
Exp249.alloc <- with(Exp249.munit.des,
                     data.frame(Lines = factor(rep(Lines, each=2), levels=1:75),
                                Checks = fac.recast(rep(Lines, each=2),
```

```
                                                  newlevels=c(rep(3, 73), 1 , 2),
                                                  levels.order = c(3,1,2),
                                                  newlabels = c("NAM","Scout","Gladius")),
                                Salt = factor(rep(1:2, times=264),
                                                  labels = c('0 NaCl','100 NaCl'))))
Exp249.recip <- fac.gen(list(Groups = 6, Cols = 4, Pairs = 11, Locations = 2))
Exp249.nest  <- list(Cols = c("Groups", "Pairs"),
                     Locations = c("Groups", "Cols", "Pairs"))
Exp249.lay <- designRandomize(allocated         = Exp249.alloc,
                              recipient         = Exp249.recip,
                              nested.recipients = Exp249.nest,
                              seed              = 51412)

#'## Add second-phase factors
#'## (to which the first-phase factors have been systematically allocated)
Exp249.lay <- cbind(fac.gen(list(Lanes = 24, Positions = 2:23)),
                    fac.gen(list(Zones = 6, Rows = 4, MainPosn = 11, Subunits = 2)),
                    Exp249.lay)

#'## Plot the assignment of Lines in the second-phase design - or see file that includes the output
Exp249.lay$Replication <- fac.recast(Exp249.lay$Lines,
                                     newlevels = rep(1:3, c(21,52,2)))
designGGPlot(Exp249.lay, labels = "Lines", cellfillcolour.column = "Replication",
             colour.values = c("lightblue", "grey", "lightgreen"),
             row.factors = "Lanes", column.factors = "Positions",
             title = "Layout of Lines for optimized design",
             reverse.x = TRUE, reverse.y = FALSE, blockdefinition = cbind(4,22))
```



5.3.2 Check the properties of the design

The maximal allocation-based mixed model is (Checks + Lines) * Salt | (Zones * MainPosn) / Rows / Subunits + (Groups * Pairs) / Cols / Locations, with Checks nested within Lines. Use the `designAnatomy` to check the properties of the design for an analysis of data from an experiment based on this design.

```
#'## Check design properties
Exp249.canon <- designAnatomy(formulae = list(carts  = ~ (Zones*MainPosn)/Rows/Subunits,
                                              tables = ~ (Groups*Pairs)/Cols/Locations,
                                              treats = ~ (Checks + Lines) * Salt),
                              data     = Exp249.lay)
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Lines[Checks] and Checks
are partially aliased in MainPosn&Pairs
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Lines[Checks] and Checks
are partially aliased in Zones#MainPosn&Groups#Pairs
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Lines[Checks] and Checks
are partially aliased in Rows[Zones:MainPosn]&Cols[Groups:Pairs]
```

```
summary(Exp249.canon)

##
##
## Summary table of the decomposition for carts, tables & treats (based on adjusted quantities)
##
##  Source.carts                   df1 Source.tables                df2 Source.treats      df3
##  Zones                            5 Groups                         5 Lines[Checks]        5
##  MainPosn                        10 Pairs                         10 Checks               2
##                                                                      Lines[Checks]        8
##  Zones#MainPosn                  50 Groups#Pairs                  50 Checks               2
##                                                                      Lines[Checks]       48
##  Rows[Zones:MainPosn]           198 Cols[Groups:Pairs]           198 Checks               2
##                                                                      Lines[Checks]       72
##                                                                      Residual           124
##  Subunits[Zones:MainPosn:Rows] 264 Locations[Groups:Pairs:Cols] 264 Salt                 1
##                                                                      Checks#Salt          2
##                                                                      Lines#Salt[Checks]  72
##                                                                      Residual           189
##  aefficiency eefficiency order
##       0.1498      0.1422     5
##       0.0033      0.0031     2
##       0.2094      0.1809     8
##       0.2111      0.2049     2
##       0.1142      0.0145    48
##       0.7854      0.7792     2
##       0.6640      0.2632    66
##       1.0000      1.0000     1
##       1.0000      1.0000     1
##       1.0000      1.0000     1
##       1.0000      1.0000     1
##       1.0000      1.0000     1
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source        df Alias  In                                    aefficiency eefficiency order
##  Lines[Checks] 2  Checks MainPosn&Pairs                            1.0000      1.0000     1
##  Lines[Checks] 2  Checks Zones#MainPosn&Groups#Pairs               1.0000      1.0000     1
##  Lines[Checks] 2  Checks Rows[Zones:MainPosn]&Cols[Groups:Pairs]    0.0944      0.0785     2
##
## The design is not orthogonal
```

Because, there is a one-to-one correspondence between the tables and carts sources, omit the tables formula and rerun — it will make the anova table more readable.

```
#'## Check design properties, with tables omitted
Exp249.canon <- designAnatomy(formulae = list(carts  = ~ (Zones*MainPosn)/Rows/Subunits,
                                              treats = ~ (Checks + Lines) * Salt),
                              data     = Exp249.lay)
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Lines[Checks] and Checks
are partially aliased in MainPosn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Lines[Checks] and Checks
are partially aliased in Zones#MainPosn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Lines[Checks] and Checks
are partially aliased in Rows[Zones:MainPosn]
```

```
summary(Exp249.canon)
```

```
##
##
## Summary table of the decomposition for carts & treats (based on adjusted quantities)
##
##  Source.carts                   df1 Source.treats      df2 aefficiency eefficiency order
##  Zones                            5 Lines[Checks]        5      0.1498      0.1422     5
##  MainPosn                        10 Checks               2      0.0033      0.0031     2
##                                     Lines[Checks]        8      0.2094      0.1809     8
##  Zones#MainPosn                  50 Checks               2      0.2111      0.2049     2
##                                     Lines[Checks]       48      0.1142      0.0145    48
##  Rows[Zones:MainPosn]           198 Checks               2      0.7854      0.7792     2
##                                     Lines[Checks]       72      0.6640      0.2632    66
##                                     Residual           124
##  Subunits[Zones:MainPosn:Rows] 264 Salt                  1      1.0000      1.0000     1
##                                     Checks#Salt          2      1.0000      1.0000     1
##                                     Lines#Salt[Checks]  72      1.0000      1.0000     1
##                                     Residual           189
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source         df Alias  In                    aefficiency eefficiency order
##  Lines[Checks] 2  Checks MainPosn                    1.0000      1.0000     1
##  Lines[Checks] 2  Checks Zones#MainPosn              1.0000      1.0000     1
##  Lines[Checks] 2  Checks Rows[Zones:MainPosn]       0.0944      0.0785     2
##
## The design is not orthogonal
```

5.3.3 Examine the properties of the design for an alternative analysis

However, rather than fit the allocation-based model, because it is known from past experience that once a linear trend for MainPosn has been fitted there are no deviations from this trend, the term xMainPosn is used to fit the trend; the term `xMainPosn` is a centred, linear covariate for MainPosn. Use the `designAnatomy` to check the properties of the design for an analysis based on a modified model, in which MainPosn in the random model has been replaced by xMainPosn in the fixed model, Zones#MainPosn has been omitted and Rows[Zones:MainPosn] has been replaced by Mainunits[Zones].

```
#'## Add factors and variates for new analysis
Exp249.lay <- within(Exp249.lay,
                     { xMainPosn <- as.numfac(MainPosn)
                       xMainPosn <- -(xMainPosn - mean(xMainPosn))
```

```
                      Mainunits <- fac.combine(list(Rows,MainPosn))
                    })
head(Exp249.lay)

##   Lanes Positions Zones Rows MainPosn Subunits Groups Cols Pairs Locations Lines Checks     Salt
## 1     1         2     1    1        1        1      1    1     1         1     3    NAM 100 NaCl
## 2     1         3     1    1        1        2      1    1     1         2     3    NAM   0 NaCl
## 3     1         4     1    1        2        1      1    1     2         1    74  Scout 100 NaCl
## 4     1         5     1    1        2        2      1    1     2         2    74  Scout   0 NaCl
## 5     1         6     1    1        3        1      1    1     3         1     1    NAM 100 NaCl
## 6     1         7     1    1        3        2      1    1     3         2     1    NAM   0 NaCl
##   Replication Mainunits xMainPosn
## 1           1         1         5
## 2           1         1         5
## 3           3         2         4
## 4           3         2         4
## 5           1         3         3
## 6           1         3         3

#'## Check properties if only linear trend fitted
set.daeTolerance(element.tol = 1e-06)
Exp249.canon <- designAnatomy(formulae = list(cart  = ~ Zones/Mainunits/Subunits,
                                              treat = ~ xMainPosn +
                                                        (Checks + Lines) * Salt),
                              data     = Exp249.lay)
summary(Exp249.canon)

##
##
## Summary table of the decomposition for cart & treat (based on adjusted quantities)
##
##  Source.cart               df1 Source.treat       df2 aefficiency eefficiency order
##  Zones                       5 Lines[Checks]        5      0.1500      0.1426     5
##  Mainunits[Zones]          258 xMainPosn            1      1.0000      1.0000     1
##                                Checks               2      1.0000      1.0000     1
##                                Lines[Checks]       72      0.9879      0.8437     6
##                                Residual           183
##  Subunits[Zones:Mainunits] 264 Salt                 1      1.0000      1.0000     1
##                                Checks#Salt          2      1.0000      1.0000     1
##                                Lines#Salt[Checks]  72      1.0000      1.0000     1
##                                Residual           189
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source        df Alias                    In    aefficiency eefficiency order
##  Checks         1 xMainPosn                treat      0.0010      0.0010     1
##  Checks         2 ## Information remaining treat      0.9995      0.9990     2
##  Lines[Checks]  1 xMainPosn                treat      0.2139      0.2139     1
##  Lines[Checks]  1 Checks                   treat      0.0003      0.0003     1
##  Lines[Checks] 72 ## Information remaining treat      0.9962      0.7859     2
##
## The design is not orthogonal
```

The Table of (partial) aliasing shows that all `treat- sources are partially aliased with \verb`xMainPosn=,

although they are not far from being orthogonal.

We have been able to check what information is available about Lines and Salt after adjustment for the linear trend. In practice, a spline term might be needed to account for nonlinearity in the trend.

5.3.4 Questions

1. What advantages accrue from randomizing Lines within Groups^Pairs (Zones^MainPosn) as compared to the original DiGGer design, in which they are randomized to Cols within Groups (Lanes within Zones) and to Pairs (MainPosn)?

   The anatomy for the `DiGGer` *design shows us that all 74 degrees of freedom are estimable in Rows#Cols[Blocks] with average efficiency 0.582 and minimum efficiency 0.209. Compared to this, the anatomy for the rerandomized design shows that the NAM lines are estimable from Rows[Zones:MainPosn], the source equivalent to Rows#Cols[Blocks], with average efficiency 0.664 and minimum efficiency 0.263. Also, the Residual degrees of freedom for Rows#Cols[Blocks] have increased from 106 degrees of freedom in the original design to 124 degrees of freedom for Rows[Zones:MainPosn] in the rerandomized design. That is, one can expect the estimation of the Lines predictions and their standard errors to be more precise for the rerandomized design.*

2. What effect does the use of a linear trend, as opposed to a set of effects, have on the analysis?

   The efficiency for Lines has increased further so that the minimum is now 0.844 and the Residual degrees of freedom for Rows[Zones:MainPosn] now stands at 183. This allows one to consider ignoring information not estimable from Rows[Zones:MainPosn], while predictions will be adjusted for the trend across MainPosn.

5.4 Two-phase, wheat experiment with 49 lines

The first, or field, phase of a wheat trial for 49 lines is laid out as an RCBD with four blocks. The produce from the field trial is processed in the second, or laboratory, phase and the design employed is a balanced lattice square for 49 treatments that involves 4 replicates each consisting of a 7×7 square. In the laboratory phase there are four intervals each of which consists of 7 runs of a machine. In a run, samples are processed at seven consecutive times. This experiment is Example 2.2 from Bailey and Brien (2016), where its anova with expected mean squares is given. Its factor-allocation diagram is in Figure 20.

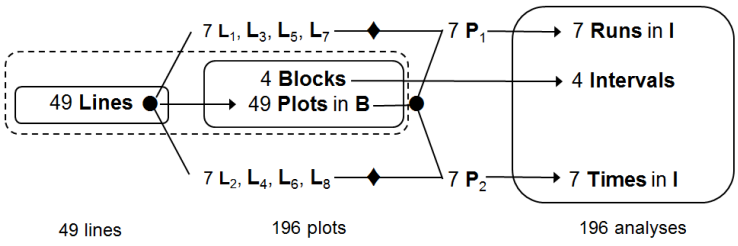
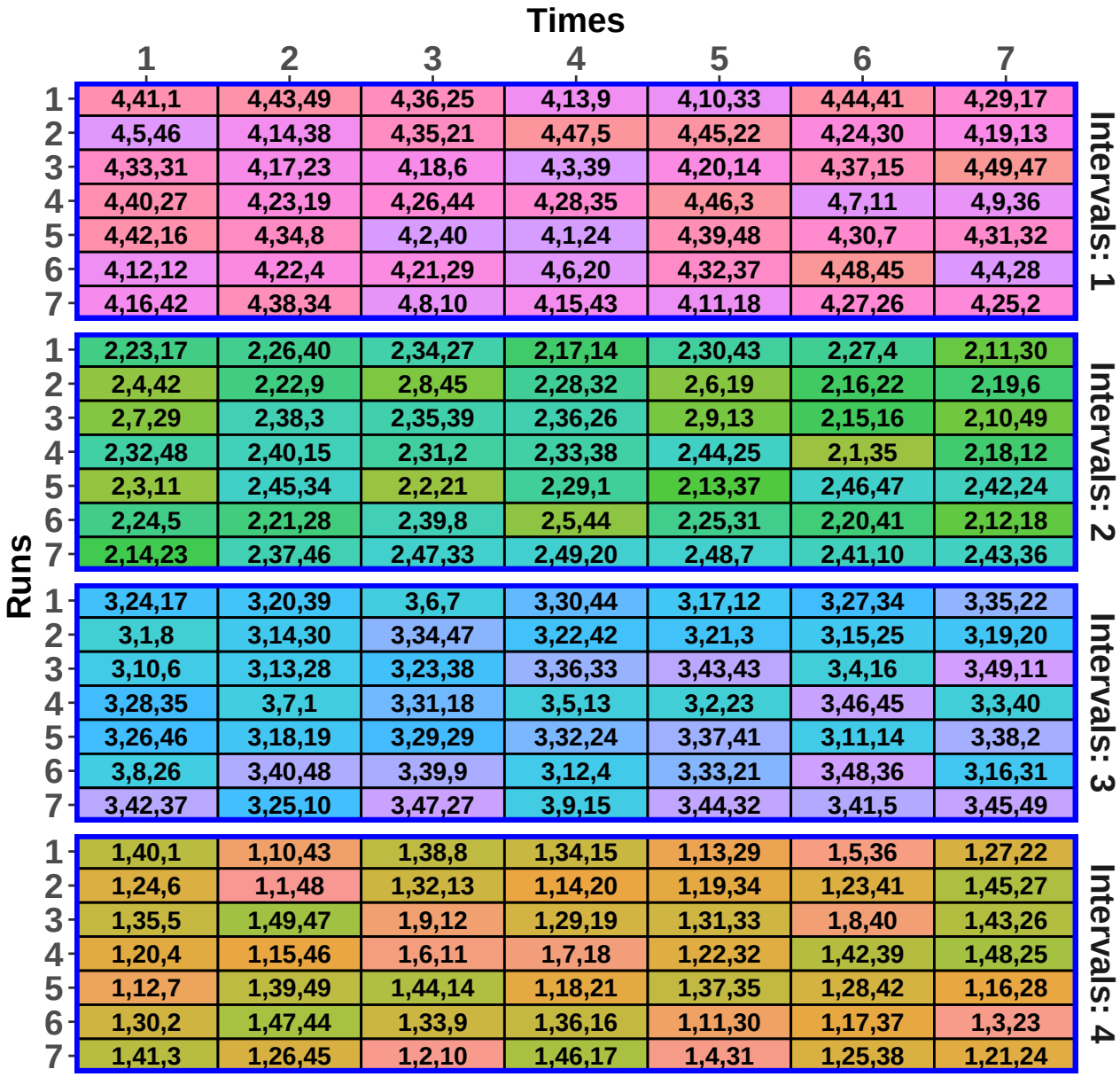


Figure 20: Factor-allocation diagram for the two-phase wheat variety experiment: lines are randomized to plots, then lines and plots are randomized to analyses; the arrow for Lines to Plots indicates that Lines are randomized to Plots; similarly, Blocks are randomized to intervals; L_1, L_3, L_5 and L_7 are pseudofactors that group the Lines for randomization to Runs and L_2, L_4, L_6 and L_8 are pseudofactors that group the Lines for randomization to Times; the two '♦' symbols indicate that the pseudofactors for Lines determine the pseudofactors P_1 and P_2 for assigning Plots to Runs and Times, respectively; B = Blocks; I = Intervals.

5.4.1 Produce randomized layout for both phases and check its properties

```
#'## Generate a layout for the field phase
field.sys <- cbind(fac.gen(list(Blocks = 4, Plots = 49)),
                   fac.gen(list(Lines = 49), times=4))
field.lay <- designRandomize(allocated         = field.sys["Lines"],
                             recipient         = field.sys[c("Blocks", "Plots")],
                             nested.recipients = list(Plots = "Blocks"),
                             seed              = 82522)
head(field.lay)

##   Blocks Plots Lines
## 1      1     1    48
## 2      1     2    10
## 3      1     3    23
## 4      1     4    31
## 5      1     5    36
## 6      1     6    11

#'## Generate laboratory phase
#'### Load a systematic balanced lattice square
data("LatticeSquare_t49.des")
#'### Form systematic design
#'### Add Intervals to field layout,  merge the data frames and sort into lab phase order
field.lay$Intervals <- field.lay$Blocks
lab.alloc <- merge(LatticeSquare_t49.des, field.lay)
lab.alloc <- with(lab.alloc, lab.alloc[order(Intervals, Runs,Times),])
lab.alloc <- lab.alloc[c("Blocks","Plots","Lines")] #Reduce columns in lab.alloc
#'### Randomize the design
lab.lay <- designRandomize(allocated         = lab.alloc,
                           recipient         = list(Intervals = 4, Runs = 7, Times = 7),
                           nested.recipients = list(Runs = "Intervals",
                                                    Times = "Intervals"),
                           seed              = 141797)
head(lab.lay)

##   Intervals Runs Times Blocks Plots Lines
## 1         1    1     1      4    41     1
## 2         1    1     2      4    43    49
## 3         1    1     3      4    36    25
## 4         1    1     4      4    13     9
## 5         1    1     5      4    10    33
## 6         1    1     6      4    44    41

#'## Plot the design to show the allocation of Blocks, Plots and Lines in the lab phase
lab.lay$FieldFactors <- with(lab.lay, fac.combine(list(Blocks, Plots, Lines),
                                                  combine.levels = TRUE))
designGGPlot(lab.lay, labels = "FieldFactors",
             row.factors = c("Intervals", "Runs"), column.factors = "Times",
             title = "Allocation of Blocks, Plots and Lines in the lab phase",
             cellalpha = 0.75, blockdefinition = cbind(7, 7))
```

Allocation of Blocks, Plots and Lines in the lab phase




```
#'## Check properties of the design
wheat.canon <- designAnatomy(formulae = list(lab    = ~ Intervals/(Runs*Times),
                                             field  = ~ Blocks/Plots,
                                             treats = ~ Lines),
                             data     = lab.lay)
summary(wheat.canon, which.criteria =c("aefficiency", "order"))

##
##
## Summary table of the decomposition for lab, field & treats (based on adjusted quantities)
##
##  Source.lab            df1 Source.field  df2 Source.treats df3 aefficiency order
##  Intervals               3 Blocks          3                         1.0000     1
```

```
##  Runs[Intervals]         24 Plots[Blocks]  24 Lines          24      0.2500     1
##  Times[Intervals]        24 Plots[Blocks]  24 Lines          24      0.2500     1
##  Runs#Times[Intervals] 144 Plots[Blocks] 144 Lines          48      0.7500     1
##                                              Residual       96      1.0000     1
##
## The design is not orthogonal
```

Given, the nonorthogonality of Blocks:Plots evident in the anatomy, redo the table with just the lab and field tiers to investigate.

```
#'## Check confounding of field sources with lab sources
wheat.labfield.canon <- designAnatomy(formulae = list(lab   = ~ Intervals/(Runs*Times),
                                                      field = ~ Blocks/Plots),
                                      data     = lab.lay)
summary(wheat.labfield.canon, which.criteria =c("aefficiency", "order"))

##
##
## Summary table of the decomposition for lab & field
##
##  Source.lab            df1 Source.field  df2 aefficiency order
##  Intervals               3 Blocks          3      1.0000     1
##  Runs[Intervals]        24 Plots[Blocks]  24      1.0000     1
##  Times[Intervals]       24 Plots[Blocks]  24      1.0000     1
##  Runs#Times[Intervals] 144 Plots[Blocks] 144      1.0000     1
```

5.4.2 Question

1. Is the variance matrix for this experiment based on two sets of terms that are orthogonal?

   Because all plots sources are confounded orthogonally with analyses sources, the variance matrix is indeed based on two sets of terms that are orthogonal.

5.5 Elaborate, two-phase, sensory experiment

Brien and Payne (1999) describe a two-phase sensory experiment, of which the first, or field, phase is a viticultural experiment and the second, or evaluation, phase involves the assessment of wine made from the produce of the first phase plots. In the field phase, two adjacent Youden squares are used to assign trellis treatments to the plots, a plot being a row-column combination within a square. Each plot is divided into two halfplots and two methods of pruning assigned at random to them. In the second phase, the halfplots from the field phase are randomized, using two Latin squares and an extended Youden design, to glasses in positions on a table for evaluation by judges. This experiment is Example 1.2 from Bailey and Brien (2016), where its anova, along with expected mean squares, is given. Its factor-allocation diagram is in Figure 21.

5.5.1 Check the properties of the randomized layout

Load the layout and use `designAnatomy` to check the properties of the design.

```
#'## Load the layout
data("Sensory3PhaseShort.dat")

#'## Examine the properties of the design
sensory.canon <- designAnatomy(formulae = list(eval   = ~((Occ/Int/Sit)*Jud)/Posn,
                                               field  = ~(Row*(Sqr/Col))/Hplot,
```

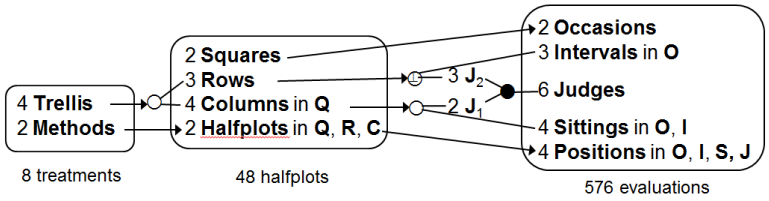


Figure 21: Factor-allocation diagram for the two-phase sensory experiment: treatments are randomized to halfplots, which are, in turn, randomized to evaluations; the arrow to the '○', the '○' and the two lines from the '○' indicate that Trellis is randomized to the combinations of Rows and Columns using a nonorthogonal design; the single arrow between Methods and Halfplots indicates that Methods is randomized to Halfplots; the single arrows between the two right hand panels indicate that Squares are randomized to Occasions and Halfpplots are randomized to Positions; J_1 and J_2 are two pseudofactors on Judges that split them into two sets of three; the Rows are randomized to the combinations of Intervals and J_2 using an orthogonal design, as indicated by the '⊕', and Columns are randomized to the combinations of J_1 and Sittings using an nonorthogonal design, as indicated by the '○'; Q= Squares; R = Rows;, C = Columns; O = Occasions; I = Intervals; S = Sittings; J = Judges.

```
                                          treats = ~Trel*Meth),
                              data     = Sensory3PhaseShort.dat)
summary(sensory.canon, which.criteria =c("aefficiency", "order"))

##
##
## Summary table of the decomposition for eval, field & treats (based on adjusted quantities)
##
##  Source.eval           df1 Source.field       df2 Source.treats df3 aefficiency order
##  Occ                     1 Sqr                  1                         1.0000     1
##  Int[Occ]                4
##  Sit[Occ:Int]           18 Col[Sqr]             6 Trel            3      0.0370     1
##                                                   Residual        3      0.3333     1
##                            Residual            12
##  Jud                     5
##  Occ#Jud                 5
##  Int#Jud[Occ]           20 Row                  2                         1.0000     1
##                            Row#Sqr              2                         1.0000     1
##                            Residual            16
##  Sit#Jud[Occ:Int]       90 Col[Sqr]             6 Trel            3      0.0741     1
##                                                   Residual        3      0.6667     1
##                            Row#Col[Sqr]        12 Trel            3      0.8889     1
##                                                   Residual        9      1.0000     1
##                            Residual            72
##  Posn[Occ:Int:Sit:Jud] 432 Hplot[Row:Sqr:Col]  24 Meth            1      1.0000     1
##                                                   Trel#Meth       3      1.0000     1
##                                                   Residual       20      1.0000     1
##                            Residual           408
##
## The design is not orthogonal
```

Note that 1/3 of Sqr:Col is partially confounded with Occ:Int:Sit and 2/3 with Occ:Int:Sit:Jud. Also, 1/9 of Trel is partially confounded with Sqr:Col and 8/9 with Row:Sqr:Col. The canonical efficiency factor for Trel in the two Sqr:Col sources is obtained by multiplying the canonical efficiency of 1/9 for Trel with that for the particular Sqr:Col source, yielding canonical efficiencies of 1/27 and 2/27.

5.5.2 Questions

1. Which is the nonorthogonal source amongst the field sources (`Source.field`) and what is its interblock and intrablock efficiency factors?

   The only nonorthogonal field source is Sqr.Col. Its interblock efficiency factor is 1/3 *and its intrablock efficiency factor is* 2/3.

2. How would an intrablock analysis be achieved using, say, regression software?

   To achieve an intrablock analysis requires careful specification of the order of fitting terms; a nonorthogonal source should not estimated until after all nonorthogonal terms with which it is confounded, except the last, have been estimated. For this experiment, terms should be fitted in the following order:

   *Occ*Jud + Row + Occ:Int/(Int + Sit) + Sqr.Col + Trel + Row:Sqr:Col + Occ:Int:Sit:Jud + Meth + Trel:Meth + Row:Sqr:Col:Hplot.*

   This will leave a Residual that corresponds to Occ:Int:Sit:Jud:Posn.

References

Bailey, R. A. (1992) Efficient semi-latin squares. *Statistica Sinica*, *2*, 413–437.

Bailey, R. A. and C. J. Brien (2016) Randomization-based models for multitiered experiments. I. A chain of randomizations. *Annals of Statistics*, *44*, 1131–1164.

Box, G. E. P., W. G. Hunter, and J. S. Hunter (2005) *Statistics for Experimenters*. (2nd ed.) New York: Wiley.

Brien, C. J. (2017) Multiphase experiments in practice: A look back. *Australian & New Zealand Journal of Statistics*, *59*(4), 327–352.

Brien, C. J. (2023a) *asremlPlus: Augments 'ASReml-R' in Fitting Mixed Models and Packages Generally in Exploring Prediction Differences*. URL http://CRAN.R-project.org/package=asremlPlus/, (R package version 4.3-50, accessed March 12, 2023).

Brien, C. J. (2023b) *dae: functions useful in the design and ANOVA of experiments*. URL http://CRAN.R-project.org/package=dae/, (R package version 3.2-15, accessed March 20, 2023).

Brien, C. J. and R. A. Bailey (2006) Multiple randomizations (with discussion). *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, *68*, 571–609.

Brien, C. J. and R. A. Bailey (2009) Decomposition tables for experiments. I. A chain of randomizations. *The Annals of Statistics*, *37*, 4184–4213.

Brien, C. J., B. Berger, H. Rabie, and M. Tester (2013) Accounting for variation in designing greenhouse experiments with special reference to greenhouses containing plants on conveyor systems. *Plant Methods*, *9*, 5. Available from http://www.plantmethods.com/content/9/1/5.

Brien, C. J. and C. G. B. Demétrio (2009) Formulating mixed models for experiments, including longitudinal experiments. *The Journal of Agricultural, Biological and Environmental Statistics*, *14*, 253–280.

Brien, C. J., B. D. Harch, R. L. Correll, and R. A. Bailey (2011) Multiphase experiments with at least one later laboratory phase. I. Orthogonal designs. *Journal of Agricultural, Biological and Environmental Statistics*, *16*, 422–450.

Brien, C. J. and R. W. Payne (1999) Tiers, structure formulae and the analysis of complicated experiments. *The Statistician*, *48*, 41–52.

Brien, C. J., R. A. Semarini, and C. G. B. Demétrio (2023) Exposing the confounding in experimental designs to understand and evaluate them, and formulating linear mixed models for analyzing the data from an experiment. *Biom. J.*,, accepted for publication.

Clingeleffer, P. R., R. S. Trayford, P. May, and C. J. Brien (1977) Use of the starwheel sprayer for applying drying emulsion to sultana grapes to be dried on the trellis. *Australian Journal of Experimental Agriculture and Animal Husbandry*, *17*, 871–880.

Cochran, W. G. and G. M. Cox (1957) *Experimental Designs*. (2nd ed.) New York: Wiley.

Coombes, N. E. (2009) *DiGGer: design search tool in R*. URL: http://nswdpibiom.org/austatgen/software/, (accessed July 16, 2017).

Gilmour, A. R., R. Thompson, and B. R. Cullis (1995) Average information reml: An efficient algorithm for variance parameter estimation in linear mixed models. *Biometrics*, *51*, 1440–1450.

Hinkelmann, K. and O. Kempthorne (2005) *Design and Analysis of Experiments*, Volume 2. of *Wiley Series in Probability and Statistics*. Hoboken, N.J.: Wiley-Interscience.

Joshi, D. D. (1987) *Linear Estimation and Design of Experiments*. New Delhi: Wiley Eastern.

McIntyre, G. A. (1955) Design and analysis of two phase experiments. *Biometrics*, *11*, 324–334.

Mead, R., S. G. Gilmour, and A. Mead (2012) *Statistical principles for the design of experiments*. Cambridge: Cambridge University Press.

R Core Team (2023) *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. URL: <http://www.r-project.org/>.

Williams, E. R., A. C. Matheson, and C. E. Harwood (2002) *Experimental Design and Analysis for Tree Improvement*. (2nd ed.) Melbourne, Australia: CSIRO.

Yates, F. (1937) The design and analysis of factorial experiments. *Imperial Bureau of Soil Science Technical Communication*, *35*.