Identifying, randomizing, canonically analyzing and formulating mixed models for designs for comparative experiments using R

(with output and solutions)

C. J. Brien

April 8, 2023

Workshop, Queensland, 18–20 April 2023

This document describes how to use functions from the R (R Core Team, 2023) packages `dae` (Brien, 2023b) and `odw` (Butler, 2022) to produce layouts for experiments and to check some of their properties. An introduction to the approach used in the document is given by Brien et al. (2023).

Contents

Topic 0 Introduction for the workshop and the software to be used

0.1 Installed software

The following software should be installed on your computer:

- R (4.1.x or later preferable)
- RStudio
- Packages (you can check the version using the `packageVersion` function.)
  - dae (Version 3.2-15 or later from CRAN (`https://cran.at.r-project.org/package=dae/`) or `http://chris.brien.name/rpackages`)
  - odw (Version 2.1.4) from `https://mmade.org/optimaldesign/`)

0.2 Programme

0.2.1 Day 1

12:00–13:00: 1. Concepts in experimental design: Experiment description, randomization by permutation based on the nesting an crossing, canonical analysis of a design and formulating allocation-based mixed models for orthogonal designs, including those with multiple errors.

13:00–13:45: Lunch.

13:45–14:30 1. (cont'd) Orthogonal experimental design in R : using dae to generate orthogonal designs for experiments.

14:30–15:30: 2. Nonorthogonal experimental design: Using the concepts in the context of balanced and unbalanced experiments; canonical efficiency factors and the alphabet of efficiency measures; the effects of covariates and missing observations.

15:30–16:00: Afternoon tea.

16:00–17:00: 2. (cont'd) Nonorthogonal experimental design in R : using dae and odw to produce nonorthogonal designs for experiments.

0.2.2 Day 2

13:00–14:00: 3. Miscellaneous topics in experimental design: systematic allocation and pseudoreplication, block-treatment interaction, designing animal grazing experiments, and nested factorials.

14:00–15:00: 3. (cont'd) Miscellaneous experimental design topics in R : further use of dae and odw.

15:00–15:15: Afternoon tea.

0.3 Packages and the functions to be used

0.3.1 dae

The package dae provides functions useful in the design and anova of experiments (Brien, 2023b). There are around 90 functions that fall into the following categories and those that will be used in this course are described:

1. Data

   BIBDWheat.dat Data for a balanced incomplete block experiment.

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

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

   Exp249.munit.des Systematic, main-unit 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

7. Miscellaneous functions.

0.3.2 odw

The package `odw` generates optimal experimental designs (Butler, 2022). It does this based on an *anticipated* mixed model and obtains a design that minimizes the average variance of pairwise differences (AVPD). It more than 30 functions; the two primary functions for this course are as follows:

odw: Generates optimal designs for comparative experiments under a general linear mixed model.

odw.options: Sets or displays various options that affect the behaviour of odw.

Documentation for each of these functions is available from the user manual for the relevant package. In general this can be found in the `doc` subdirectory of the directory in which the package is installed or from the `help` for the function once the package has been installed. For the latter, to see the manual for package `foo`, enter `help(package="foo")` and click on the link `User guides, package vignettes and other documentation`.

For `dae`, the manual is available via `vignette("dae-manual", package="dae")` and there are some notes that show how to use the functions that are available via `vignette("DesignNotes", package="dae")`.

0.4 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 residual variation are underlined.

Topic 1 Orthogonal experimental design in R

This class of experiments covers the orthogonal standard or textbook experiments, those that 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, an allocated set that is allocated to a recipient set. These two sets are also referred to as 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] '3.2.17'
```

1.1 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., 2023) for the RCBD is in Figure 1 and that for the LSqD is in Figure 2.

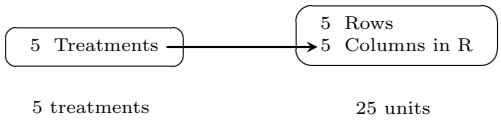


Figure 1: Factor-allocation diagram for an RCBD: treatments are allocated to units; the arrow indicates that the factor Treatments is randomized to Columns; Columns in R indicates that the Columns are considered to be nested within Rows for this randomization; R = Rows.

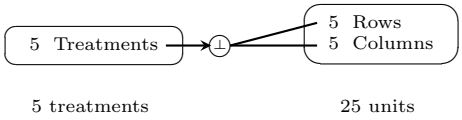


Figure 2: Factor-allocation diagram for an LSqD: treatments are allocated to units; the arrow indicates that the allocation is randomized; the '⊥' at the end of the arrow indicates that an orthogonal design is used; the two lines from '⊥' indicates that the Treatments are allocated to the combinations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

1.1.1 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) `recipient`, (iii) `nested.recipients`, 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.

Our 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 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, in these notes, 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 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.

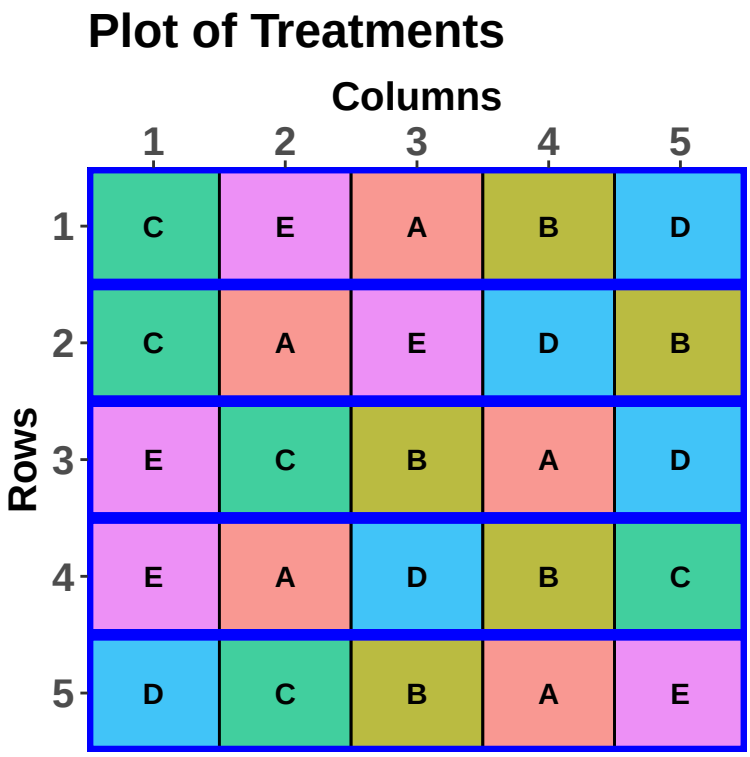
In this example, the allocated factor is Treatments, with 5 levels, and the recipient factors are Rows and Columns, both with 5 levels. Suppose that Rows are to form the blocks.

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

```
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 randomized 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 argument settings.

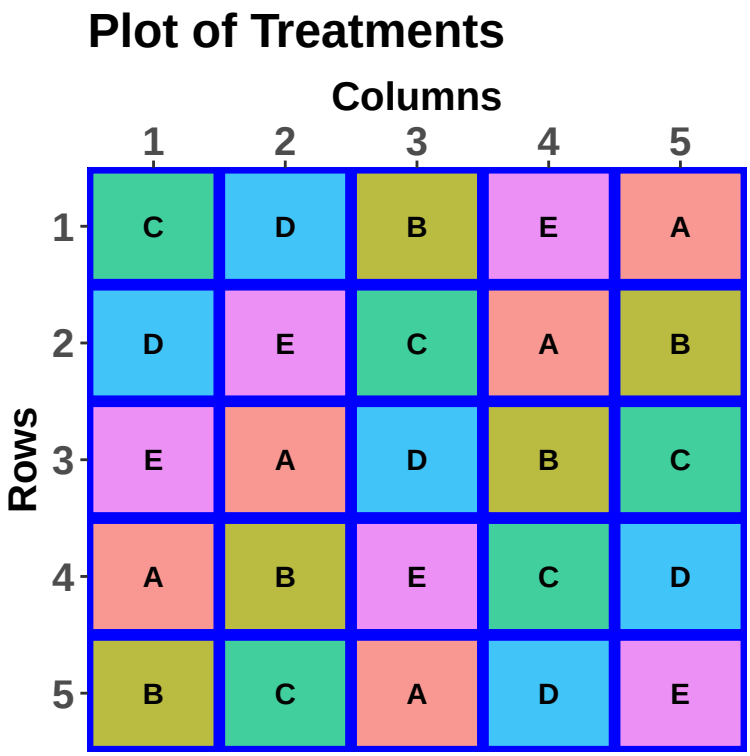
1.1.2 Produce the randomized layout for an LSqD

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 so that `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 randomized 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))
```



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.

1.1.3 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 (like skeleton-anova tables but produced from an eigenanalysis) 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`.

```
#'## Get the 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
```

1.1.4 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 between the designs in the interpretation of these units 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.

1.2 Split-unit from Kaps and Lamberson (2004)

Kaps and Lamberson (2004, p.344) describes a split-unit experiment that investigates the effects of four different pasture treatments and two mineral supplements on the milk yield of cows. The Pasture treatments are assigned to the main units formed from large plots using a randomized complete-block design with 3 blocks and the Mineral supplements are randomly assigned to the subunits (smaller plots) in each main unit. The factor-allocation diagram for the experiment is in Figure 3.

1.2.1 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:

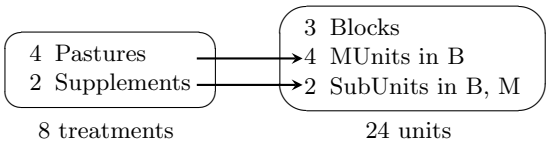
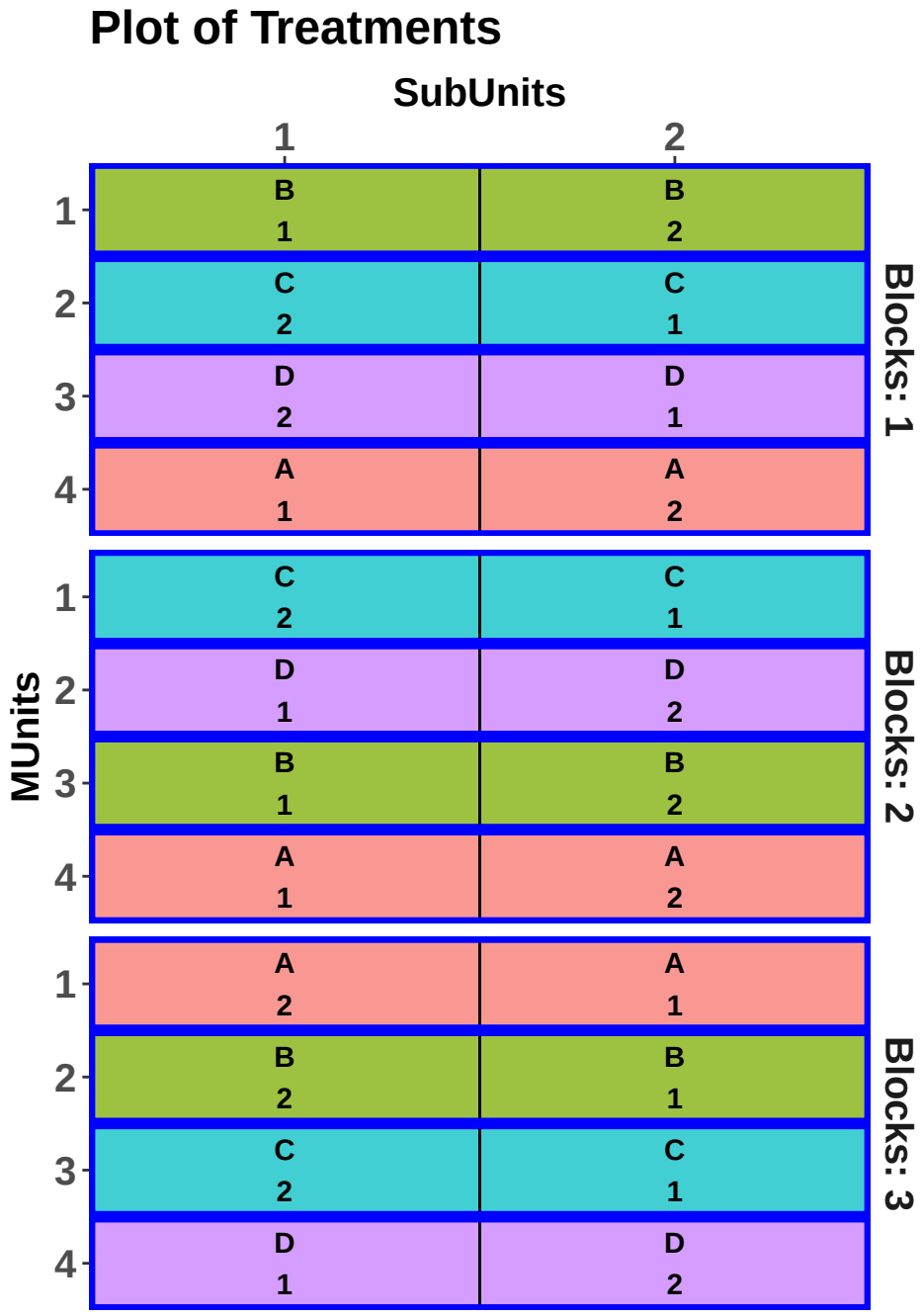


Figure 3: Factor-allocation diagram for a split-plot design: treatments are allocated to units; the arrows indicates that the factors Pastures and Supplements are randomized to MUnits and SubUnits, respectively; MUnits in B indicates that the MUnits are considered to be nested within Blocks for this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

```
#'## Set up the systematic design
Milk.sys <- cbind(fac.gen(list(Blocks=3, MUnits=4, SubUnits=2)),
                  fac.gen(list(Pastures=LETTERS[1:4],
                               Supplements=2), times=3))

#'## Obtain the randomized layout
Milk.lay <- designRandomize(allocated         = Milk.sys[c("Pastures", "Supplements")],
                            recipient         = Milk.sys[c("Blocks", "MUnits", "SubUnits")],
                            nested.recipients = list(MUnits = "Blocks",
                                                     SubUnits = c("MUnits", "Blocks")),
                            seed              = 580523)
#'## Plot design produced, first combining Pastures and Supplements so plot on 2 lines per cell
Milk.lay$Treatments <- with(Milk.lay, fac.combine(list(Pastures, Supplements),
                                                  combine.levels = TRUE, sep = "\n"))
designGGPlot(Milk.lay, labels = "Treatments",
             row.factors = c("Blocks", "MUnits"), column.factors = "SubUnits",
             cellfillcolour.column = "Pastures", cellalpha = 0.75,
             blockdefinition = rbind(c(1,2)))
```




```
#'## Check its properties
Milk.canon <- designAnatomy(formulae = list(unit = ~ Blocks/MUnits/SubUnits,
                                            trt  = ~ Pastures*Supplements),
                            data     = Milk.lay)
summary(Milk.canon, which.criteria = c("aeff", "order"))

##
##
## Summary table of the decomposition for unit & trt
##
##  Source.unit             df1 Source.trt           df2 aefficiency order
```

```
##  Blocks                    2
##  MUnits[Blocks]            9 Pastures              3      1.0000     1
##                              Residual              6
##  SubUnits[Blocks:MUnits]  12 Supplements           1      1.0000     1
##                              Pastures#Supplements  3      1.0000     1
##                              Residual              8
```

1.2.2 Questions

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

   In the sense that the randomization of both Supplements and Pastures can be achieved with a single permutation of the units, the subunits.

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

   The initial allocation mixed model is Pastures + Supplements + Pastures:Supplements | Blocks + Blocks:MUnits + Blocks:MUnits:SubUnits. The initial allocation model is equivalent to a randomization model because all allocation was by randomization.

3. A factorial RCBD would involve randomizing the $3 \times 4 = 12$ treatments to the 12 subunits within each block. What is the effect on treatment comparisons of using the split-unit design as compared to a factorial RCBD?

   The precision of the Pastures differences may be less than the precision of the Supplements differences, depending on the how much extra variablity there is between MUnits as compared to the variability between SubUnits. If there is extra variation between the MUnits as compared to the SubUnit, then the Residual mean square for MUnits[Blocks] will be larger than that for SubUnits[Blocks:MUnits]. If a factorial RCBD had been used, the Residual mean square for Units[Blocks] would be the weighted average of the two Residual mean squares from the split-unit experiment, the weights being their 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-unit design. Consequently, the comparison between Supplements within Varieties will be more precise for the split-unit design.

1.3 Split-unit design with criss-cross design on the subunits from Mead (1990)

Mead (1990, Example 14.1) describes an experiment to investigate the effects of grazing patterns on pasture composition. It is available in `dae` as `SPLGrass.dat`.

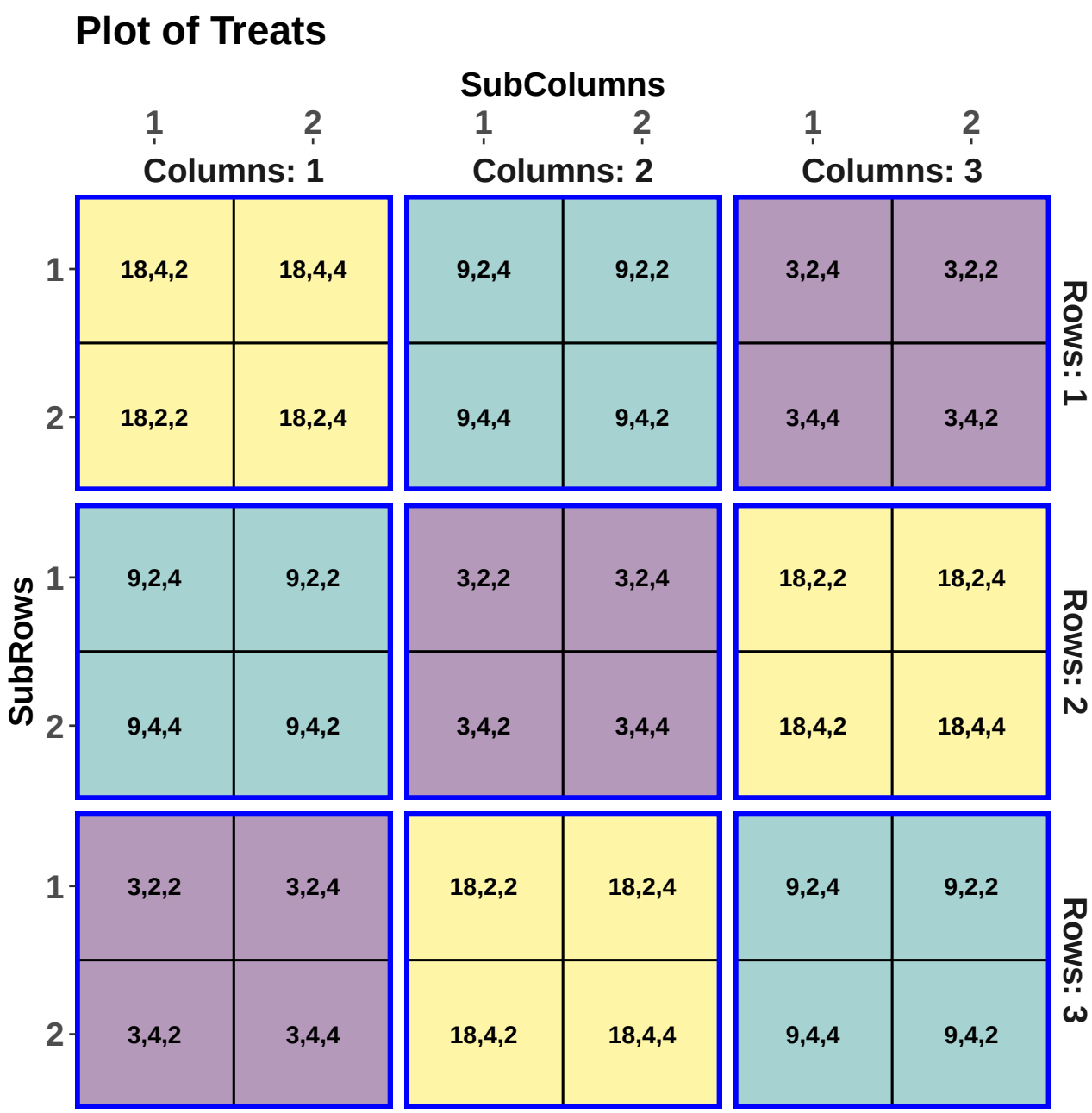
The design for the experiment is a split-unit design with a criss-cross or strip-unit design on the subunits. The main units are arranged in 3 Rows $\times$ 3 Columns. Each main unit is split into 2 SubRows x 2 SubColumns.

The factor Period, with levels 3, 9 and 18 days, is assigned to the main units using a 3×3 Latin square. The two-level factors Spring and Summer are assigned to subunits using a criss-cross or strip-unit design that is randomized within each main unit; Spring is randomized to SubRows and Summer is randomized to SubColumns. The levels of each of Spring and Summer are two different grazing patterns in its season. The response variable is `Main.Grass`.

Use `data(SPLGrass.dat)` to load the design (and the data). Plot the design: create a factor Treats by combining the factors Period, Spring and Summer using `fac.combine` with the argument `combine.levels` set to `TRUE` and then use `designGGPlot` with `cellfillcolour.column` set to `"Period"` and `cellalpha` set to about 0.4. Also, investigate the properties of the design using `designAnatomy`.

```
#'## Load the design
data("SPLGrass.dat")

#'## Plot the design
SPLGrass.dat$Treats <- with(SPLGrass.dat,
                            fac.combine(list(Period, Spring, Summer), combine.levels = TRUE))
```

```
designGGPlot(SPLGrass.dat, labels = "Treats",
             row.factors = c("Rows", "SubRows"), column.factors = c("Columns", "SubColumns"),
             cellfillcolour.column = "Period", cellalpha = 0.4,
             blockdefinition = cbind(2,2))
```




```
#'## Check its properties
Grass.canon <- designAnatomy(formulae = list(unit = ~ (Rows*Columns)/(SubRows*SubColumns),
                                             trt  = ~ Period*Spring*Summer),
                             data     = SPLGrass.dat)
summary(Grass.canon, which.criteria = c("aeff", "order"))

##
```

```
##
## Summary table of the decomposition for unit & trt
##
##  Source.unit                        df1 Source.trt           df2 aefficiency order
##  Rows                                 2
##  Columns                              2
##  Rows#Columns                         4 Period                 2      1.0000     1
##                                         Residual               2
##  SubRows[Rows:Columns]                9 Spring                 1      1.0000     1
##                                         Period#Spring          2      1.0000     1
##                                         Residual               6
##  SubColumns[Rows:Columns]             9 Summer                 1      1.0000     1
##                                         Period#Summer          2      1.0000     1
##                                         Residual               6
##  SubRows#SubColumns[Rows:Columns]     9 Spring#Summer          1      1.0000     1
##                                         Period#Spring#Summer   2      1.0000     1
##                                         Residual               6
```

1.3.1 Questions

1. Describe the confounding that is inherent in this design.

   Period is confounded with Rows#Columns; Spring and Period#Spring are confounded with SubRows[Rows:Columns], while Summer and Period#Summer are confounded with SubColumns[Rows:Columns]. Finally Spring#Summer and Period#Spring#Summer are confounded with SubRows#SubColumns[Rows:Columns].

2. Draw a factor-allocation diagram for this experiment.

   You should have (i) a treatments panel with 3 Periods, 2 Spring and 2 Summers, (ii) a plots panel with 3 Rows, 3 Columns, 2 SubRows in R, C, 2 SubColumns in R, C. There should be an arrow from Periods to an orthogonal design symbol and two lines from the symbol to Rows and Columns, as well as arrows from Spring to SubRows and Summer to SubColumns.

3. What is the initial allocated mixed model for this design?

   The initial allocation mixed model is Period + Spring + Period:Spring + Summer + Period:Summer + Spring:Summer + Spring:Summer | Rows + Columns + Rows:Columns + Rows:Columns:SubRows + Rows:Columns:SubColumns + Rows:Columns:SubRows:SubColumns. The initial allocation model is equivalent to a randomization model because the allocation was only by randomization.

1.4 A design for the 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 for allocating the 4 Additives to the 32 tests, being the combinations of the 2 Occasions x 4 Drivers x 4 Cars. Make sure that a Latin square is used for each Occasion.

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.

```
#'## 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"),
```

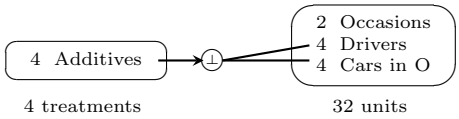
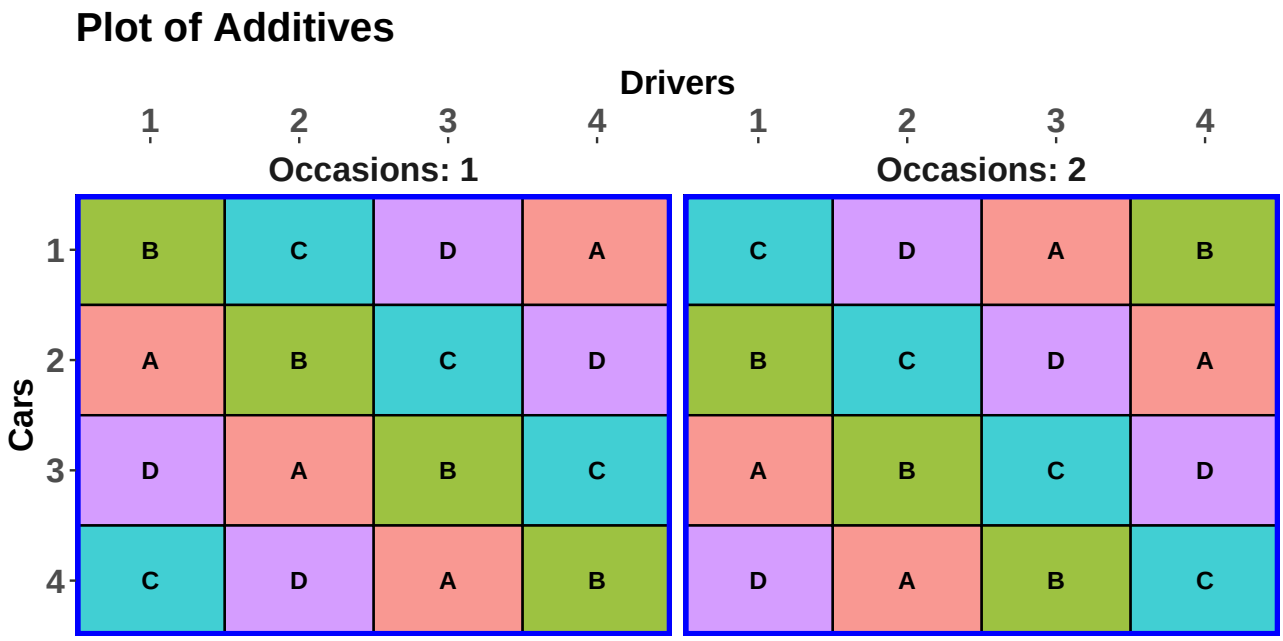


Figure 4: Factor-allocation diagram for repeated LSqDs: treatments are allocated to units; the arrow indicates that the allocation is randomized; the '⊥' at the end of the arrow indicates that an orthogonal design is used; the two lines from '⊥' indicates that the Additives are allocated to the combinations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

```
                                seed              = 194)

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




```
#'## Get the anatomy of the layout
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
```

Secondly, 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.

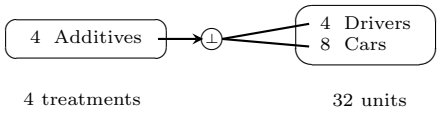
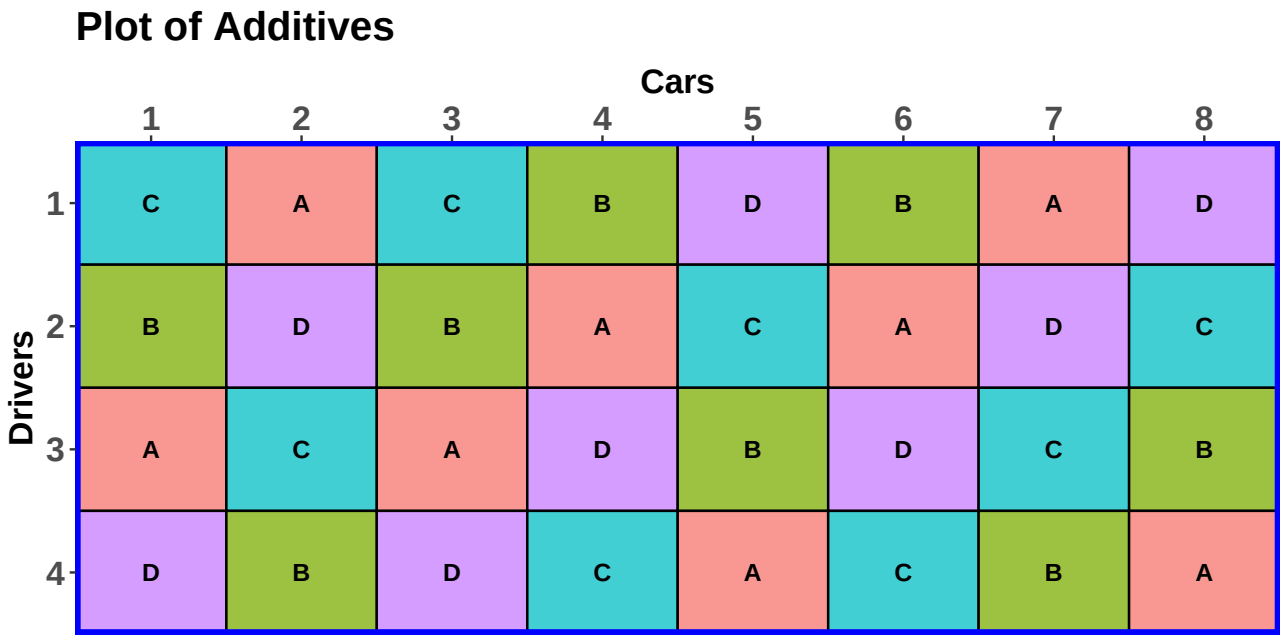


Figure 5: Factor-allocation diagram for repeated LSqDs: treatments are allocated to units; the arrow indicates that the allocation is randomized; the '①' at the end of the arrow indicates that an orthogonal design is used; the two lines from '①' indicates that the Additives are allocated to the combinations of Drivers and Cars using the design.

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

1.4.1 Questions

1. The Residual degrees of freedom for a single 4×4 Latin square are 6. Has the use of two 4×4 Latin squares had the desired effect of increasing the Residual df? What other advantage does the use of two Latin squares have over the use of a single Latin square?

   Yes, the Residual df have been increased from 6 to 15. Using two Latin squares doubles the replication as compared to a single Latin square, thereby increasing the precision of the experiment by decreasing the standard error of differences between pairs of Additive means.

2. 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.

3. 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].

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

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

Topic 2 Nonorthogonal experimental design in R

This class of experiments covers the nonorthogonal standard or textbook experiments and these experiments must be single phase because they involve a single randomization.

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

```
library(dae, quietly = TRUE)
library(odw)

## Loading required package:  Matrix

packageVersion("odw")

## [1] '2.1.4'

options(width=100)
```

2.1 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.

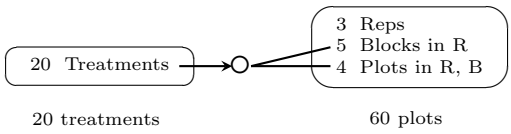


Figure 6: Factor-allocation diagram for the alpha design: treatments are allocated to units; the arrow indicates that the allocation is randomized; the 'O' at the end of the arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicate that the Treatments are allocated to the combinations 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.

2.1.1 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 (A), that is, to 0.7447. In particular, $AVPD = 2/(rA)$.

```
AVPD <- designAmeasures(mat.Vpredicts(target = ~ Treats - 1,
                                      fixed  = ~ Reps/Blocks,
                                      design = alpha.lay))[[1]]
Aeff <- summary(alpha.canon, which.criteria = "aeff")$decomp$aefficiency[3]
(measures <- c(AVPD, Aeff, 2/(3*Aeff)))

## [1] 0.8952381 0.7446809 0.8952381
```

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
```

2.1.2 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).

2.2 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.

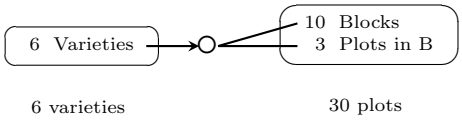


Figure 7: Factor-allocation diagram for the balanced incomplete-block design: treatments are allocated to units; the arrow indicates that the allocation is randomized; the 'O' at the end of the arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicates that the Varieties are 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.

2.2.1 Load the design and check its 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.

Calculate the AVPD and check that $AVPD = 2/(rA)$

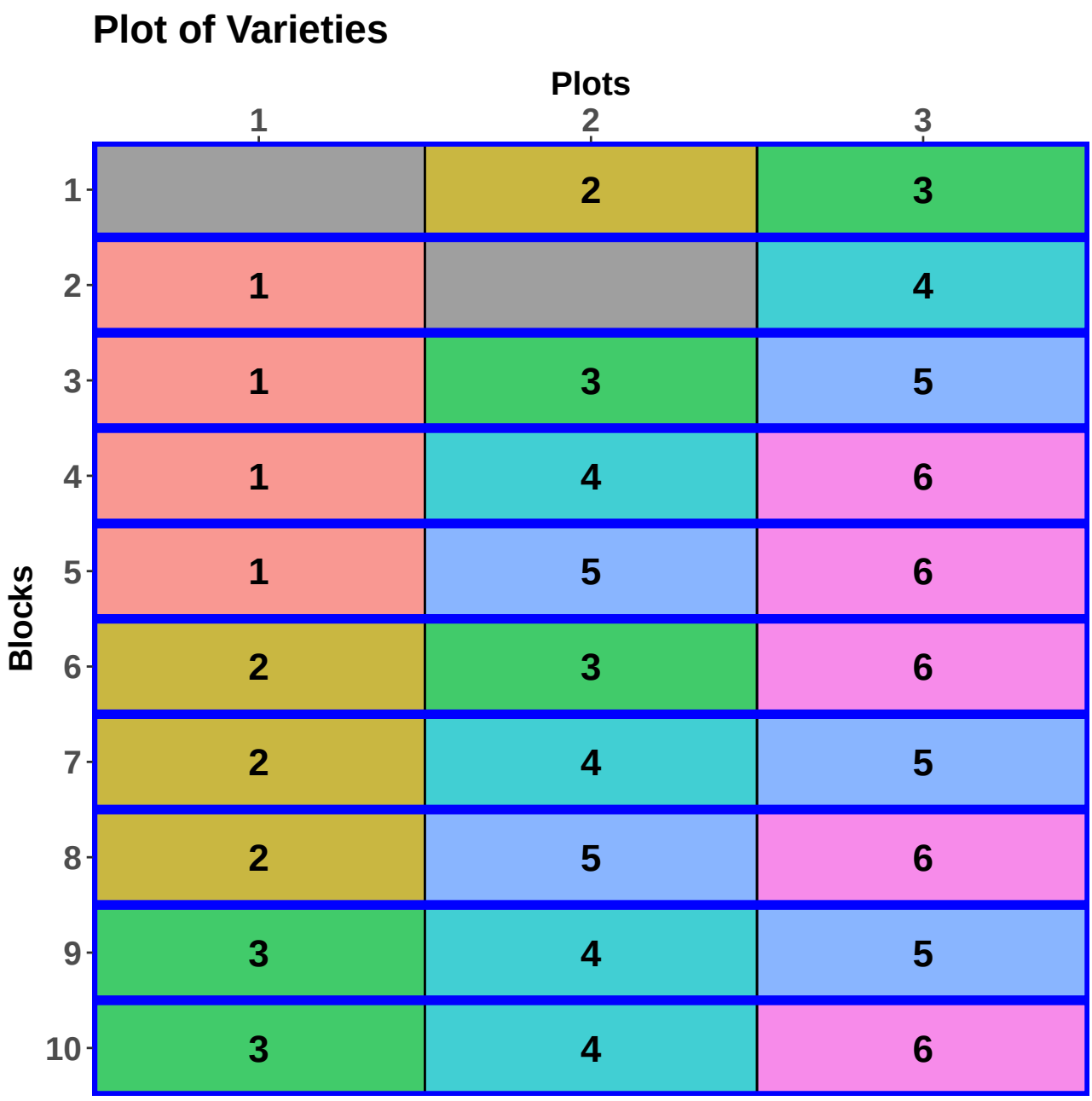
```
AVPD <- designAmeasures(mat.Vpredicts(target = ~ Varieties - 1,
                                      fixed  = ~ Blocks,
                                      design = BIBDWheat.dat))[[1]]
Aeff <- summary(bibdwheat.canon, which.criteria = "aeff")$decomp$aefficiency[3]
(measures <- c(AVPD, Aeff, 2/(5*Aeff)))

## [1] 0.5 0.8 0.5
```

2.2.2 What if two observations are missing?

Set the two observations that are not the Control to missing and obtain the anatomy The greatest effect is surprisingly on the comparison between the Control and New.

```
#'## Investigate the effect of two-missing observations
#+ "BIBDDet"
bibdwheat.Miss.dat <- BIBDWheat.dat
```

```
bibdwheat.Miss.dat$Varieties[c(1,5)] <- NA #different Blocks & Varieties
designGGPlot(bibdwheat.Miss.dat, labels = "Varieties",
             row.factors = "Blocks", column.factors = "Plots",
             cellalpha = 0.75, size = 6, blockdefinition = cbind(1,3))

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




```
bibdwheat.Miss.canon <- designAnatomy(formulae = list(units = ~ Blocks/Plots,
                                                      trts  = ~ Varieties),
                                      data     = na.omit(bibdwheat.Miss.dat))
summary(bibdwheat.Miss.canon, which.criteria = c('aeff', "xeff", "eeff", 'order'))

##
```

```
##
## Summary table of the decomposition for units & trts (based on adjusted quantities)
##
##  Source.units  df1 Source.trts df2 aefficiency xefficiency eefficiency order
##  Blocks          9 Varieties     5      0.1909      0.4365      0.1154     5
##                    Residual      4
##  Plots[Blocks]  18 Varieties     5      0.7513      0.8846      0.5635     5
##                    Residual     13
##
## The design is not orthogonal
```

2.2.3 Questions

1. What is the value of xefficiency for Varieties when confounded with Plots[Blocks] for the original 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. What is the effect of the missing values on the efficiency for Varieties when confounded with Plots[Blocks]?

   There are now 5 different canonical efficiency factor ranging from 0.56 to 0.88 with an average of 0.75. This compares with all values equal to 0.80 for the full design.

2.3 A design with rows and columns for a Casuarina trial

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.

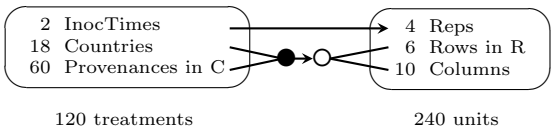


Figure 8: Factor-allocation diagram for the balancved lattice design: treatments are allocated to units; the arrows indicates that the allocations are randomized; the two lines leading to the '●' indicate that it is is the combinations of Countries and Provenances that is allocated; the '○' at the end 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.

2.3.1 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.

2.3.2 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.

2.4 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.

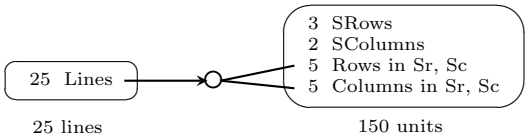


Figure 9: Factor-allocation diagram for the balanced lattice square design: treatments are allocated to units; the arrows indicates that the allocations are randomized; the 'O' at the end of the lower arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicates that the Lines 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.

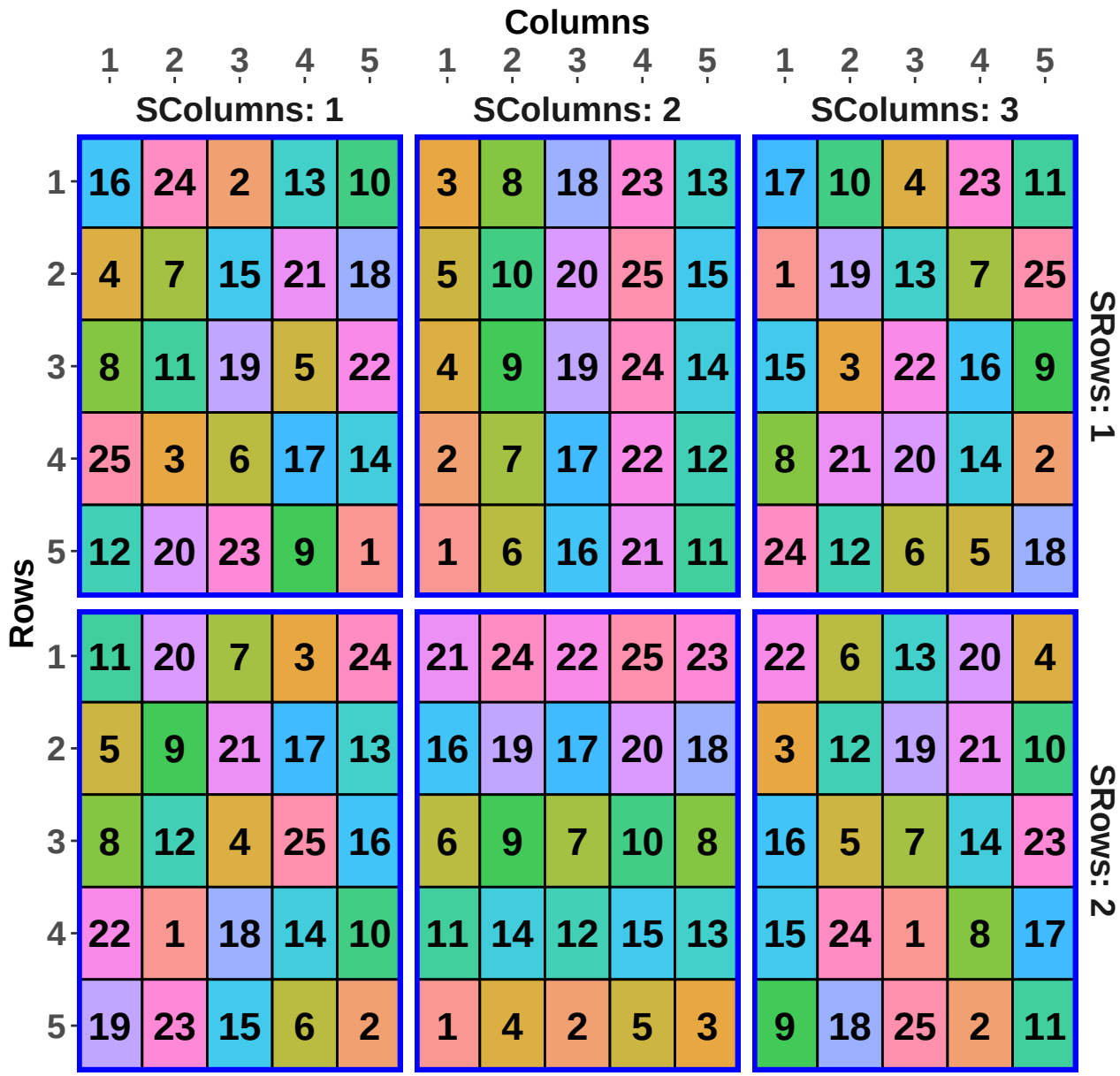
In the lectures it was stated that, while the design is optimal for a fixed model, it is not optimal for a mixed model. In this exercise, a search will be made for a resolved design that is near-A-optimal under a mixed model.

2.4.1 Input the design and check the properties of the design

Use the following R code to input and extract the design, plot it and check its properties. The R package `asremlPlus` Brien (2023a) can be used to access the data set or it is available in an `rda` data file. Because we are going to use the design to produce a new design for another experiment, we rerandomize the design using

the randomization appropriate to the balanced lattice square design. Being a valid randomization in that it corresponds to the randomization model, the properties of the design will be unchanged.

```
#'## 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"])

#'## Rerandomize the design for a new experiment
latt.lay <- designRandomize(allocated = latt.lay["Variety"],
                            recipient = latt.lay[c("SRows", "Rows", "SColumns", "Columns")],
                            nested.recipients = list(Rows = "SRows",
                                                     Columns = "SColumns"),
                            seed = 63146)

#'## Add row and column factors that have a unique level for each row and each column (needed for ar1)
latt.lay <- cbind(fac.gen(list(ARows = 10, AColumns = 15)),latt.lay)

#'## 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),
                                            trt   = ~ Variety),
                            data     = latt.lay)
summary(latt.canon, which.criteria = c("aeff", "order"))

##
##
## Summary table of the decomposition for units & trt (based on adjusted quantities)
##
##  Source.units                 df1 Source.trt 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
```

2.4.2 Search for a near-A-optimal design

Use `odw` to search for a near-A-optimal design under a mixed model for a crossed row-column design with autocorrelations, as opposed to a nested row-column design with independent errors. In this case the "tabu+rw" search method is to be used. Further, the `odw` options are to be set to values that I have found by trail-and-error to be successful. The options are

P: the probability of accepting a non-improving design; the default is P=0.005.

localSearch: the number of steps in the random walk local search strategy of the "tabu+rw" search option; the default is 10000.

tabuStop: if the number of consecutive tabu loops with no change in the objective function exceeds tabuStop, then tabu optimization terminates (the default is 4).

```
#'## Set odw options
maxit <- 25
search <- "tabu+rw"
odw.options(P = 0.10, localSearch = 10000, tabuStop = 100)
#'## Set up the values of the variance components and autocorrelation for the random terms
params <- c(2.5, 1, 0.1, 0.1, 0.5, 1, 0.6, 0.4)
names(params) <- c("g.sRR", "g.sCC", "g.sRsCR", "g.sRsCC", "g.u", "g.aRaC", "rho.R", "rho.C")
#'### Set the values in odw
Wheat.start <- odw(fixed    = ~ SRows*SColumns + Variety,
                   random   = ~ SRows:Rows + SColumns:Columns +
                                SRows:SColumns:(Rows + Columns) + units,
                   residual = ~ ar1(ARows):ar1(AColumns),
                   permute  = ~ Variety, swap = ~ SRows:SColumns,
                   data     = latt.lay, start.values = TRUE)
vp.table <- Wheat.start$vparameters.table
vp.table$Value <- params
print(vp.table)

##                      Component Value
## 1                    SRows:Rows   2.5
## 2             SColumns:Columns   1.0
## 3          SRows:SColumns:Rows   0.1
## 4       SRows:SColumns:Columns   0.1
## 5                        units   0.5
## 6             ARows:AColumns!R   1.0
## 7    ARows:AColumns!ARows!cor   0.6
## 8 ARows:AColumns!AColumns!cor   0.4

#'### Generate the near-A-optimal design
Wheat.odw <- odw(fixed    = ~ SRows*SColumns + Variety,
                 random   = ~ SRows:Rows + SColumns:Columns +
                              SRows:SColumns:(Rows + Columns) + units,
                 residual = ~ ar1(ARows):ar1(AColumns),
```

```
                 permute  = ~ Variety, swap = ~ SRows:SColumns,
                 G.param  = vp.table, R.param = vp.table,
                 maxit    = maxit, search = search,
                 data     = latt.lay)

## Sat Apr 08 19:12:12 2023
## Initial criterion = 0.385054 (25 A-equations; rank C 24)
## Criterion after 1000 initial random iterations: 0.378096
## Criterion after tabu loop 1 is 0.372342
## Criterion after tabu loop 2 is 0.372342
## Criterion after tabu loop 3 is 0.371882
## Criterion after tabu loop 4 is 0.371851
## Criterion after tabu loop 5 is 0.371821
## Criterion after tabu loop 6 is 0.371821
## Criterion after tabu loop 7 is 0.371821
## Criterion after tabu loop 8 is 0.371769
## Criterion after tabu loop 9 is 0.371769
## Criterion after tabu loop 10 is 0.371769
## Criterion after tabu loop 11 is 0.371769
## Criterion after tabu loop 12 is 0.371517
## Criterion after tabu loop 13 is 0.371517
## Criterion after tabu loop 14 is 0.371480
## Criterion after tabu loop 15 is 0.371480
## Criterion after tabu loop 16 is 0.371392
## Criterion after tabu loop 17 is 0.371233
## Criterion after tabu loop 18 is 0.371189
## Criterion after tabu loop 19 is 0.371189
## Criterion after tabu loop 20 is 0.371189
## Criterion after tabu loop 21 is 0.371189
## Criterion after tabu loop 22 is 0.371162
## Criterion after tabu loop 23 is 0.371162
## Criterion after tabu loop 24 is 0.371162
## Criterion after tabu loop 25 is 0.371162
## Hash table size 422
## Final criterion after 25 tabu+rw iterations: 0.371162
## Cleaning up: Sat Apr 08 19:12:32 2023

Wheat.lay <- Wheat.odw$design
Wheat.lay$unit <- factor(1:nrow(Wheat.lay))
```

Given that this is a spatial design, it cannot be now randomized. However, the initial design from which it was derived was randomized, thereby guarding against systematic patterns that might have been artefacts from a systematic input design.

2.4.3 Checking the properties of the designs

Now calculate the A-measure for the original lattice-square design and the near-optimal design produce by `odw`. Also, produce the anatomy for the near-optimal design.

```
#'### Calculate the A-measure for the lattice square design under a mixed model
latt.lay$unit <- factor(1:nrow(latt.lay))
(A.latt <- designAmeasures(mat.Vpredicts(target = ~ Variety - 1,
                                         fixed  = ~ SRows*SColumns - 1,
                                         random = ~ SRows:Rows + SColumns:Columns +
```

```
                                                SRows:SColumns:(Rows + Columns) + unit - 1,
                                        G      = as.list(params[1:5]),
                                        R      = kronecker(mat.ar1(params["rho.R"], 10),
                                                           mat.ar1(params["rho.C"], 15)),
                                        design = latt.lay))[[1]])

## [1] 0.3850544

#'### Check the A-value for the near-optimal design
(A.wht <- designAmeasures(mat.Vpredicts(target = ~ Variety - 1,
                                        fixed  = ~ SRows*SColumns - 1,
                                        random = ~ SRows:Rows + SColumns:Columns +
                                                   SRows:SColumns:(Rows + Columns) + unit - 1,
                                        G      = as.list(params[1:5]),
                                        R      = kronecker(mat.ar1(params["rho.R"], 10),
                                                           mat.ar1(params["rho.C"], 15)),
                                        design = Wheat.lay))[[1]])

## [1] 0.3711616

(A.wht/A.latt)

## [1] 0.9639197

#'## Check the properties of the design
Wheat.canon <- designAnatomy(formulae = list(unit = ~ (SRows:SColumns)/(Rows*Columns),
                                             trt  = ~ Variety),
                             data     = Wheat.lay)
summary(Wheat.canon, which.criteria = c("aeff", "meff", "xeff", "eeff", "order"))

##
##
## Summary table of the decomposition for unit & trt (based on adjusted quantities)
##
##  Source.unit                  df1 Source.trt df2 aefficiency mefficiency xefficiency eefficiency
##  SRows:SColumns                 5
##  Rows[SRows:SColumns]          24 Variety     23      0.0023      0.1739      0.4926      0.0001
##                                   Residual     1
##  Columns[SRows:SColumns]       24 Variety     24      0.0002      0.1667      0.4984      0.0000
##  Rows#Columns[SRows:SColumns]  96 Variety     24      0.6321      0.6667      0.9137      0.4033
##                                   Residual    72
##  order
##
##     23
##
##     24
##     24
##
##
## The design is not orthogonal
```

2.4.4 Questions

1. How do the AVPD values calculated by `odw` and those calculated using `designAmeasures` and `mat.Vpredicts` compare?

   They are the same.

2. Summarize the differences between the original balanced lattice square design and the `odw` design. Is the increased precision of the `odw` design worthwhile?

   The AVPD has decreased by around 3% and so the increase in precision is small. The lattice square design is balanced, the order of Lines always being one, and so all contrasts have equal variance. On the other hand, for the odw *design, Lines has order 24, the same as the number of degrees of freedom. The values of the efficiencies range from 0.4249 to 0.9335 so that the variances of the contrast will vary. It seems that the balance of the lattice square design is not worth sacrificing for the minor increase in precision. However, this is for the values of the variance parameters used in the call to* odw*. It would be safest to conduct a study of the value obtained for a range of values for the variance parameters.*

Topic 3 Miscellaneous experimental design topics in R

This section includes examples covering the recognition pseudoreplication, grazing trials 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)
library(odw)
packageVersion("odw")

## [1] '2.1.4'

options(width=100)
```

3.1 An animal feeding experiment

Suppose an animal scientist wants to investigate the effect on the weight gain of calves fed four different feed mixtures. They have four pens available for the experiment and they randomize the mixtures to these pens. Each pen has six calves and the weight gain of the each calf is obtained. The factor-allocation diagram for the experiment is in Figure 10.

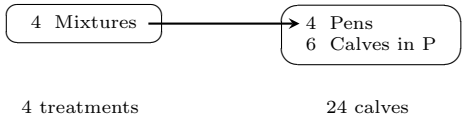
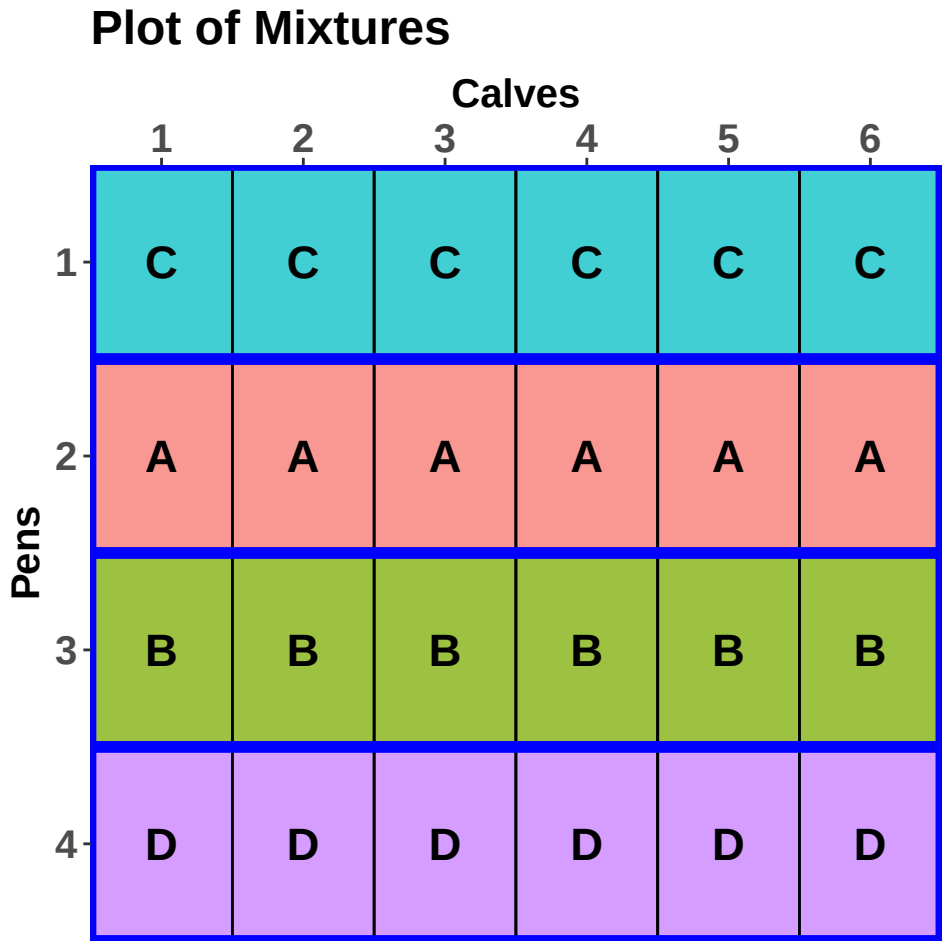


Figure 10: Factor-allocation diagram for the animal feeding experiment: treatments are allocated to calves; the arrow indicates that the factor Mixtures is randomized to Pens; Calves in P indicates that the Calves are nested within Pens; P = Pens.

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

```
#'## Set up the systematic design and obtain the layout
Feed.sys <- cbind(fac.gen(list(Pens=4, Calves=6)),
                  Mixtures = factor(rep(LETTERS[1:4], each=6)))
Feed.lay <- designRandomize(allocated         = Feed.sys["Mixtures"],
                            recipient         = Feed.sys[c("Pens", "Calves")],
                            nested.recipients = list(Calves = "Pens"),
                            seed              = 872159)

#'## plot the design
designGGPlot(Feed.lay, labels = "Mixtures",
             row.factors = "Pens", column.factors = "Calves",
             cellalpha = 0.75, size = 6, blockdefinition = cbind(1,6))
```




```
#'## Check its properties
Feed.canon <- designAnatomy(formulae = list(unit = ~Pens/Calves,
                                            trt  = ~Mixtures),
                            data     = Feed.lay)
summary(Feed.canon)

##
##
## Summary table of the decomposition for unit & trt
##
##  Source.unit  df1 Source.trt df2 aefficiency eefficiency order
##  Pens           3 Mixtures     3      1.0000      1.0000     1
##  Calves[Pens]  20
```

3.1.1 Questions

1. How is the pseudoreplication involved in this experiment manifested in the anatomy?

   Because (i) Pens and Mixtures 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 Pens and Mixtures are inextricably confounded. That is, the pseudoreplication has resulted in differences between Pens and between Mixtures being completely mixed up.

2. The randomization-based mixed model for the experiment is Mixtures | Pens + Pens:Calves. 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 Pens is confounded with Mixtures. A fit could be obtained by removing Pens from the random model. The problem is that a test of Mixtures would then be based on the ratio of variability in Mixtures differences to an estimate of the variance of Calves-within-Pens variability. This does not include Pens 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 Pens variability is included in the denominator of the F-statistic.

3.2 Grazing experiments

Consider an experiment in which weaners are to be fed on of the three pasture regimens. The pasture regimens are to be assigned to plot in a field using a generalized randomized block design that has four blocks, each with six plots. There are 24 weaners to be assigned one to a plot.

Obtain the randomized layout for a design in which the weaners are divided into four Classes of six Weaners each, based on initial weight, and the Classes are to be assigned to the Blocks and the Weaners within a Class are to be assigned to the Plots within a Block, as described in Kaps and Lamberson (2004, p. 280–1).

```
n <- 24  #number of weaners
b <- 4   #number of blocks
t <- 3   #number of treatments
a <- 2   #number of weaners per block-treatment

#'### Generate a systematic GRBD for assigning treatments to plots
GRBD.sys <- cbind(fac.gen(list(Blocks = b, Plots = t*a)),
                  fac.gen(list(Regimens = t, a), times = b))

#'### Randomize treatments to plots
GRBD.lay <- designRandomize(recipient = GRBD.sys[c("Blocks", "Plots")],
                            allocated = GRBD.sys["Regimens"],
                            nested.recipients = list(Plots = "Blocks"),
                            seed = 158211)

#'### Generate animals factors
Animal.C2B.sys <- fac.gen(list(Classes = b, Weaners = t*a))

#'### Randomize the plots and treatments to animals
GRBD.C2B.lay <- designRandomize(recipient = Animal.C2B.sys,
                                allocated = GRBD.lay,
                                nested.recipients = list(Weaners = "Classes"),
                                seed = 82572)

#'### Plot the layout
#+ WeanerGRBD_C2B
GRBD.C2B.lay$TreatClass <- with(GRBD.C2B.lay, fac.combine(list(Regimens, Classes),
                                                          combine.levels = TRUE))
designGGPlot(GRBD.C2B.lay, labels = "TreatClass",  label.size = 6,
             title = "Plot of Regimens, Classes",
             row.factors = "Plots", column.factors = "Blocks",
             cellfillcolour.column = "Regimens", cellalpha = 0.75,
```

```
            blockdefinition = cbind(t*a, 1))
```

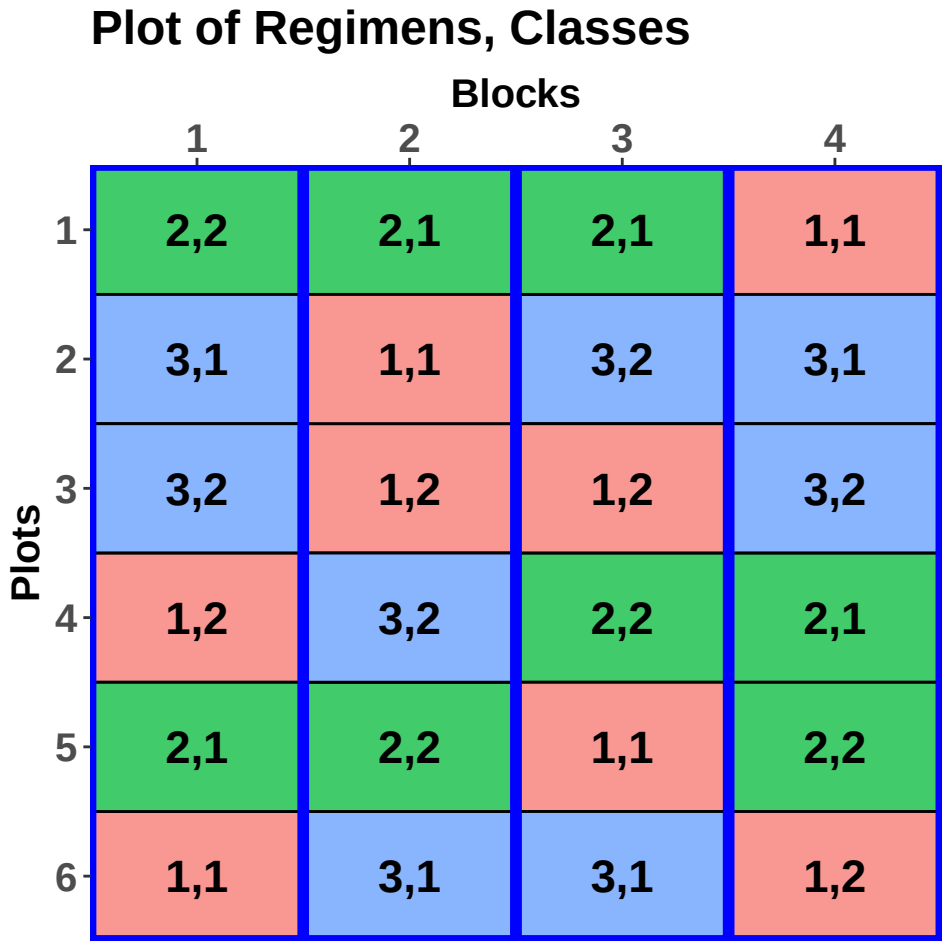
Check the properties of the layout using an anatomy.

```
#'### Check the anatomy
GRBD.C2B.canon <- designAnatomy(formula = list(anim = ~ Classes/Weaners,
                                               plot = ~ Blocks/Plots,
                                               trt  = ~ Regimens*(Classes+Blocks)),
                                data = GRBD.C2B.lay)

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

summary(GRBD.C2B.canon, which.criteria = c("aeff", "order"))

##
##
## Summary table of the decomposition for anim, plot & trt (based on adjusted quantities)
##
##  Source.anim      df1 Source.plot   df2 Source.trt       df3 aefficiency order
##  Classes            3 Blocks          3 Classes            3      1.0000     1
##  Weaners[Classes]  20 Plots[Blocks]  20 Regimens          2      1.0000     1
```

```
##                                       Regimens#Classes   6      1.0000     1
##                                       Residual          12      1.0000     1
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source          df Alias            In  aefficiency order
##  Blocks          3  Classes          trt      1.0000     1
##  Blocks          0  ## Aliased       trt      1.0000     1
##  Regimens#Blocks 6  Regimens#Classes trt      1.0000     1
##  Regimens#Blocks 0  ## Aliased       trt      1.0000     1
```

Obtain a second randomized layout for the grazing experiment in which two Weaners, one from each of two Classes based on initial weight, are assigned to the two plots within each Block that received the same Regimen, as discussed by Roberts (1975) and Brien and Demétrio (1998). Use `GRBD.sys` for the field desifn from the first grazing experiment,

```
#'### Generate animals factors, generate pseudofactors and randomize P2
Animal.C2P.sys <- fac.gen(list(Weaners = b*t, Classes = a))
GRBD.lay <- cbind(with(GRBD.lay, GRBD.lay[order(Blocks,Regimens),]),
                  fac.gen(list(P1 = t, P2 = a), each = b))
GRBD.lay <- designRandomize(allocated = GRBD.lay[c("Plots", "Regimens")],
                            recipient = GRBD.lay[c("Blocks","P1","P2")],
                            nested.recipients = list(P2 = c("Blocks", "P1")),
                            except = c("Blocks", "P1"),
                            seed = 75415)
GRBD.lay <- GRBD.lay[c("Blocks", "Plots", "Regimens")]

#'### Randomize treatments and plots to animals (using reordered GRBD)
GRBD.C2P.lay <- designRandomize(recipient = Animal.C2P.sys,
                                allocated = GRBD.lay,
                                nested.recipients = list(Weaners = "Classes"),
                                seed = 158211)

#'### Plot the layout
#+ WeanerGRBD_C2P
GRBD.C2P.lay$TreatClass <- with(GRBD.C2P.lay, fac.combine(list(Regimens, Classes),
                                                          combine.levels = TRUE))
designGGPlot(GRBD.C2P.lay, labels = "TreatClass", label.size = 6,
             title = "Plot of Regimens, Classes",
             row.factors = "Plots", column.factors = "Blocks",
             cellfillcolour.column = "Regimens", cellalpha = 0.75,
             blockdefinition = cbind(t*a, 1))
```




```
#'### Check the properties using an anatomy
GRBD.C2P.canon <- designAnatomy(formula = list(anim = ~ Classes/Weaners,
                                               plot = ~ Blocks/Plots,
                                               trt  = ~ Regimens*(Classes+Blocks)),
                                data = GRBD.C2P.lay)
summary(GRBD.C2P.canon, which.criteria = c("aeff", "order"))

##
##
## Summary table of the decomposition for anim, plot & trt
##
##  Source.anim      df1 Source.plot   df2 Source.trt       df3 aefficiency order
##  Classes            1 Plots[Blocks]   1 Classes            1      1.0000     1
##  Weaners[Classes]  22 Blocks         3 Blocks             3      1.0000     1
##                       Plots[Blocks]  19 Regimens           2      1.0000     1
##                                         Regimens#Classes   2      1.0000     1
##                                         Regimens#Blocks    6      1.0000     1
##                                         Residual           9      1.0000     1
```

3.2.1 Questions

1. How is the assignment of Classes to Plots or Blocks achieved in the R code?

It is determined by the alignment of the systematic animal factors (Classes and Anminals) aligns with the factors in the GRBD layout.

2. How does the aliasing reported in the output arise in the case in which Classes are randomized to Blocks?

   In an attempt to estimate the block-treatment interactions, both Blocks and Classes are included in the third formula, named 'trts'. However, the decomposition table shows that Classes and Blocks are inextricably confounded and so including them together in the same formula manifests as the two sources being aliased.

3. What advantages does assigning Classes to Plots have over assigning Classes to Blocks? Are there any disadvantages?

```
## Error:  '\#' is an unrecognized escape in character string starting ""\n\n\\emph{The main
advantage is that it is possible to separate the two block-treatment interactions, Regimens\#"
```

3.3 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.

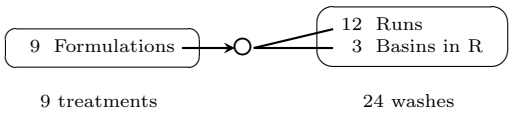


Figure 11: Factor-allocation diagram for the detergent experiment: treatments are allocated to washes; the arrow indicates that the allocation is randomized; the 'O' at the end of the arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicate that the Treatments are allocated 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.

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

```
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,
```

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

```
                                        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
```

3.3.2 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
```

3.3.3 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
```

3.3.4 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.

3.4 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.

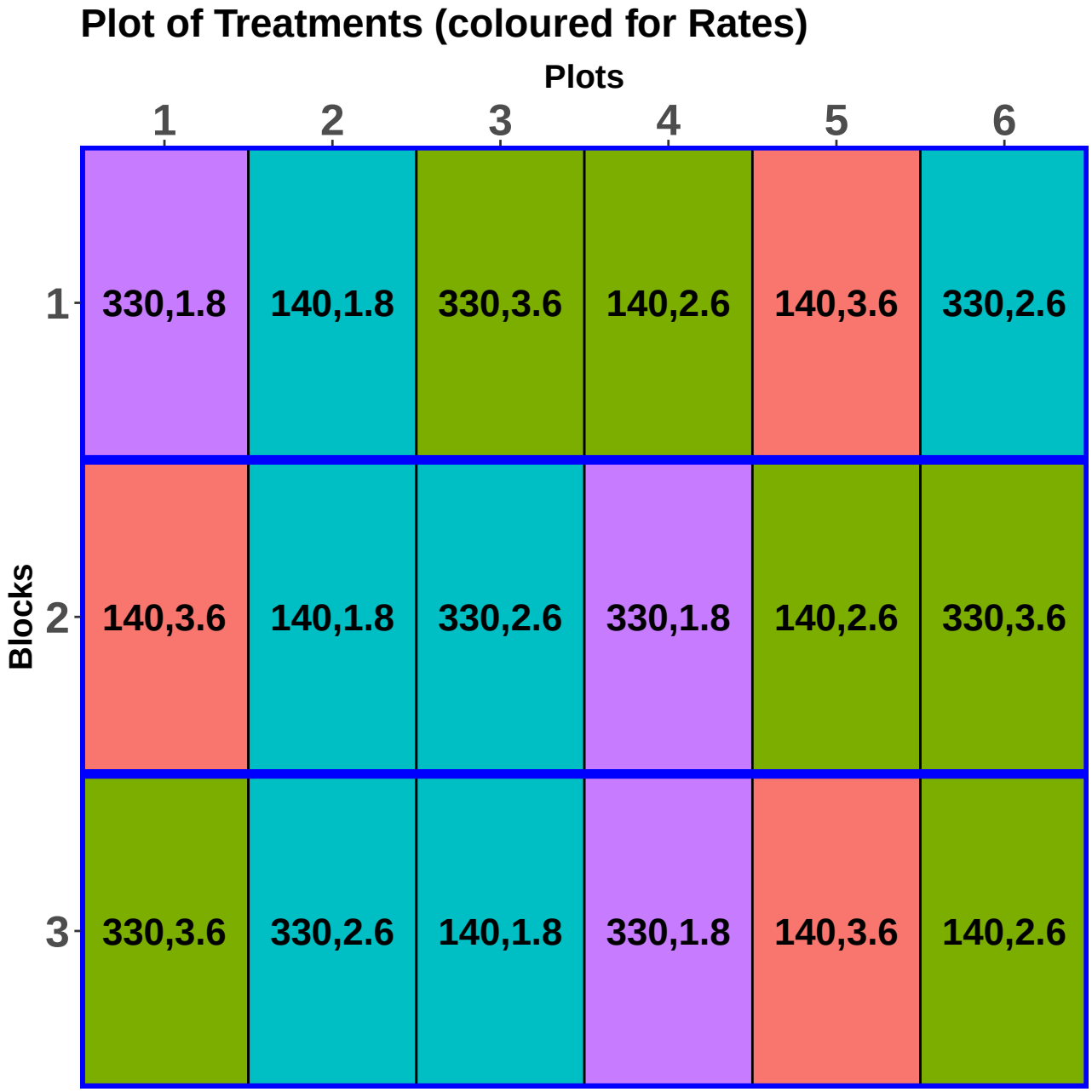
We set up the RCBD for Speed and Pressure then derive the Rate factors.

```
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
```

3.4.1 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.

3.5 A Control treatment for an incomplete-block design

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

Input the design.

```
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),
                                       labels = LETTERS[1:t]))
```

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

#'### 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 odw 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)

## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Treatments[Control] and
Control are partially aliased in Blocks
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Treatments[Control] and
Control are partially aliased in Units[Blocks]

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

##
##
## Summary table of the decomposition for unit & trt (based on adjusted quantities)
##
##  Source.unit   df1 Source.trt          df2 aefficiency xefficiency eefficiency order dforthog
##  Blocks          5 Control               1     0.1000      0.1000      0.1000     1        0
##                    Treatments[Control]   1     0.2500      0.2500      0.2500     1        0
##                    Residual              3
##  Units[Blocks]  18 Control               1     0.9000      0.9000      0.9000     1        0
##                    Treatments[Control]   4     0.8824      1.0000      0.7500     3        2
##                    Residual             13
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source              df Alias   In            aefficiency xefficiency eefficiency order dforthog
##  Treatments[Control] 1  Control Blocks             1.0000      1.0000      1.0000     1        1
##  Treatments[Control] 1  Control Units[Blocks]      0.0196      0.0196      0.0196     1        0
##
## 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
```

Now use `odw`, to obtain a near-A-optimal under a fixed model using a randomization of the treatment to the plots within incomplete blocks for the initial design.

```
#'### Initialize with a randomized layout
PBIBD.ini <- cbind(fac.gen(list(Blocks=b, Units=k)),
                   Treatments = factor(rep(1:t, times = b*k/t), labels = LETTERS[1:t]))
PBIBD.ini <- designRandomize(allocated         = PBIBD.ini["Treatments"],
                             recipient         = PBIBD.ini[c("Blocks", "Units")],
                             nested.recipients = list(Units = "Blocks"),
                             seed              = 4794)

#'### Get the odw design for fixed Blocks
PBIBD.odw <- odw(fixed   = ~ Blocks + Treatments,
                 permute = ~ Treatments,
                 search  = "tabu", maxit = 25,
                 data    = PBIBD.ini)
```

```
## Sat Apr 08 19:12:35 2023
## Initial criterion = 0.566667 (6 A-equations; rank C 5)
## Criterion after 1000 initial random iterations: 0.559487
## Criterion after tabu loop 1 is 0.559487
## Criterion after tabu loop 2 is 0.559487
## Criterion after tabu loop 3 is 0.559487
## Criterion after tabu loop 4 is 0.559487
## Criterion after tabu loop 5 is 0.559487
## Criterion after tabu loop 6 is 0.559487
## Criterion after tabu loop 7 is 0.559487
## Criterion after tabu loop 8 is 0.559487
## Criterion after tabu loop 9 is 0.559487
## Criterion after tabu loop 10 is 0.559487
## Criterion after tabu loop 11 is 0.559487
## Criterion after tabu loop 12 is 0.559487
## Criterion after tabu loop 13 is 0.559487
## Criterion after tabu loop 14 is 0.559487
## Criterion after tabu loop 15 is 0.559487
## Criterion after tabu loop 16 is 0.559487
## Criterion after tabu loop 17 is 0.559487
## Criterion after tabu loop 18 is 0.559487
## Criterion after tabu loop 19 is 0.559487
## Criterion after tabu loop 20 is 0.559487
## Criterion after tabu loop 21 is 0.559487
## Criterion after tabu loop 22 is 0.559487
## Criterion after tabu loop 23 is 0.559487
## Criterion after tabu loop 24 is 0.559487
## Criterion after tabu loop 25 is 0.559487
## Hash table size 1
## Final criterion after 25 tabu iterations: 0.559487
## Cleaning up: Sat Apr 08 19:12:36 2023

PBIBD.odw.lay <- PBIBD.odw$design
```

Randomize the design obtained using `odw` and check its properties

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

##    Blocks Units Treatments
## 1       1     1          B
## 2       1     2          F
## 3       1     3          D
## 4       1     4          E
## 5       2     1          F
## 6       2     2          C
## 7       2     3          A
## 8       2     4          E
## 9       3     1          F
## 10      3     2          D
```

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

```
#'### Check properties of the odw layout
PBIBD.odw.canon <- designAnatomy(formulae = list(plots = ~ Blocks/Units,
                                                 trts  = ~ Treatments),
                                 data      = PBIBD.odw.lay)
summary(PBIBD.odw.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    4      0.0937      0.1875      0.0625     2        0
##                    Residual      1
##  Units[Blocks]  18 Treatments    5      0.8937      1.0000      0.8125     3        1
##                    Residual     13
##
## The design is not orthogonal
```

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.

2. How do the Cochran and Cox design and the design obtained with odw compare?

   The aefficiency of the Cochran and Cox design is less than that for the odw design. However, the Cochran and Cox design has only two different efficiency factors (0.75 and 1) and has 3 orthogonal degrees of freedom. This compares with 3 efficiency factors and 1 orthogonal degree of freedom for the odw design. Sacrificing approximately 0.01 in aefficiency to have a design that is closer to balanced seems acceptable.

3.6 The Casuarina experiment (continued)

In Section 2.3 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 among 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
```

3.6.1 Questions

1. How does this analysis compare with that conducted in Section 2.3?

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

Topic 4 Using R for advanced experimental design

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

```
library(dae)
library(odw)
options(width=100)
```

4.1 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.

4.1.1 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.

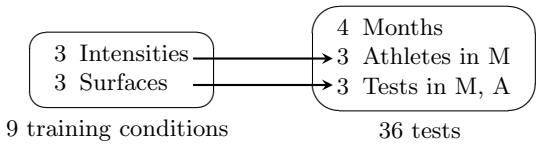
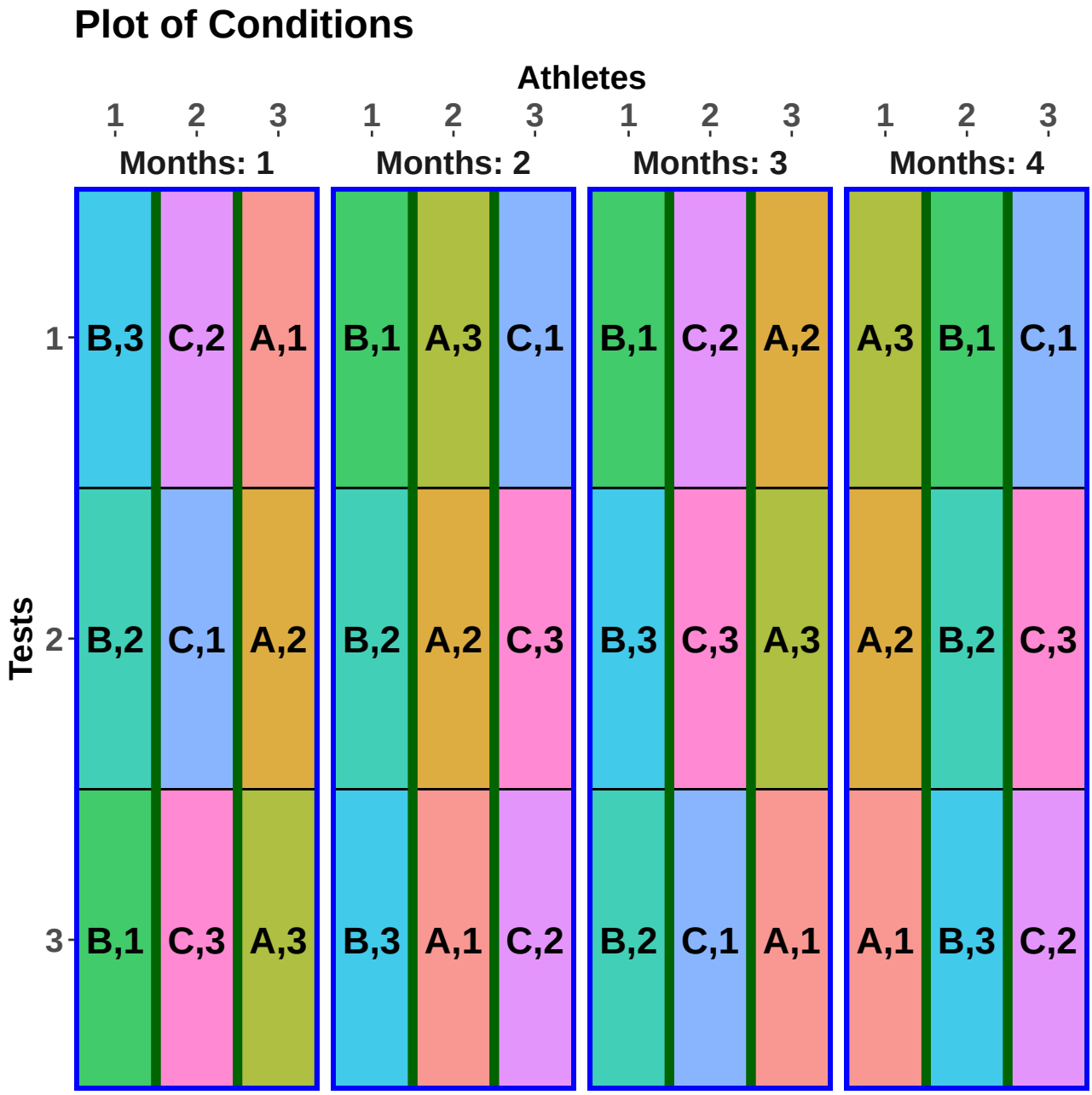


Figure 12: Factor-allocation diagram for the standard athlete training experiment: training conditions are randomized to tests; the two left-hand arrows indicate that the levels of Intensities and Surfaces are randomized to Athletes and Tests, respectively; M = Months; A = Athletes.

```
#'## 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)

#'## 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)))
```




```
#'## 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
```

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.

4.1.2 A simple two-phase athlete training experiment

Multiphase experiments differ from those previously presented in that they 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`.

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 combined-units design.

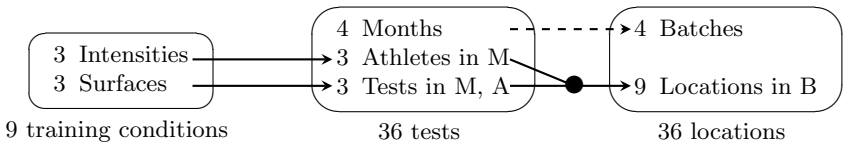


Figure 13: Factor-allocation diagram for the two-phase athlete training experiment: training conditions are randomized to tests and tests are allocated to locations; the two left-hand arrows indicate that the levels of Intensities and Surfaces are randomized to Athletes and Tests, respectively; the dashed 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.

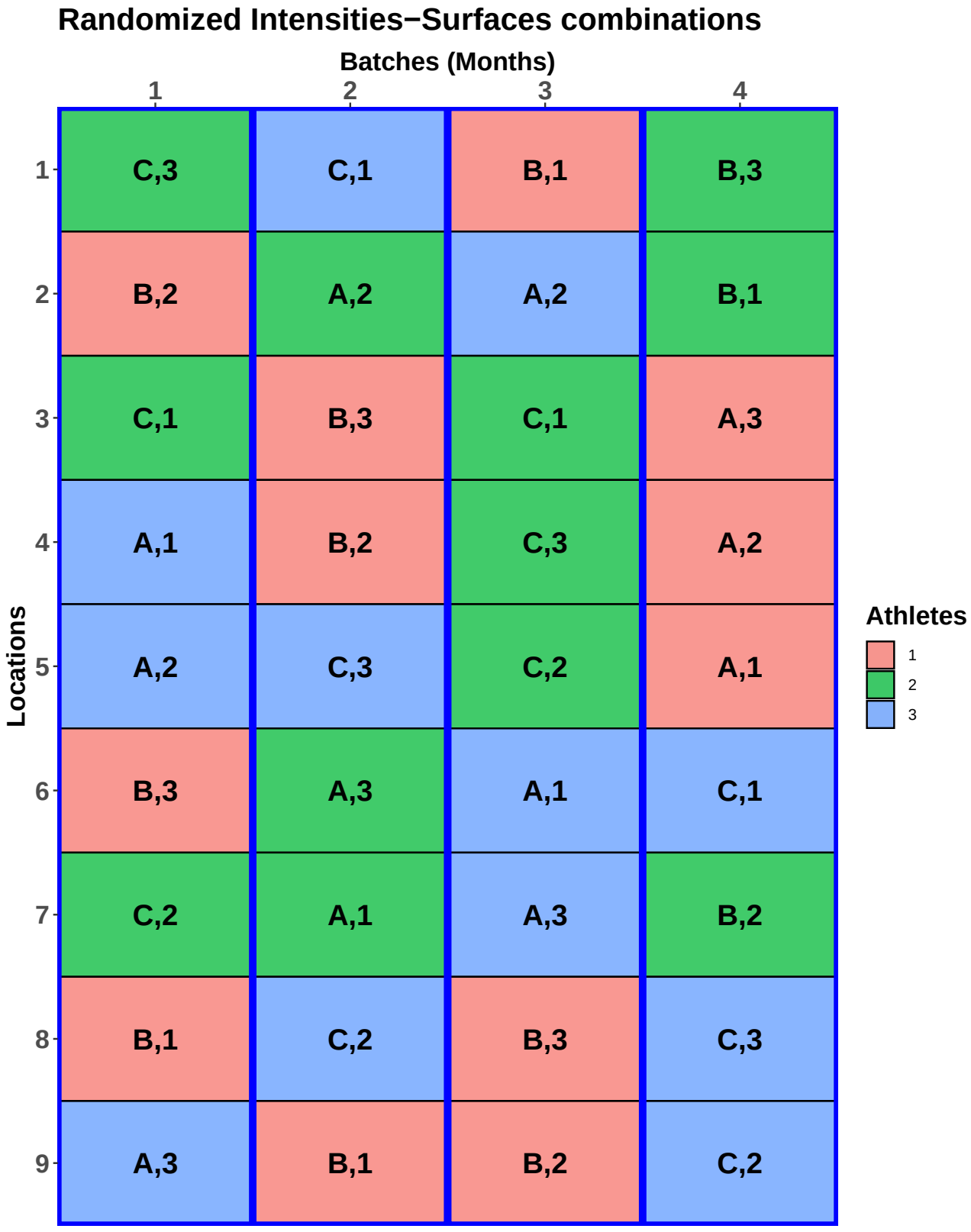
```
#'# Generate a layout for a simple two-phase athlete training experiment
#'
#'## Phase 1 - the split-plot design that has already been generated.
#'## Phase 2 - randomize tests (and training conditions) to locations,
```

```
#'##           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.

4.1.3 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 4.1.2: it assumes that lab processing order within a batch is important and so the second-phase are now crossed; hence a row-column design is required for this phase. 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 combined-units 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 combined-units 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 genotypes go from the training conditions sources to the genotypes 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 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.).

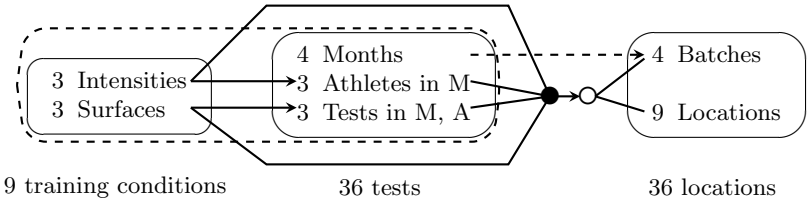
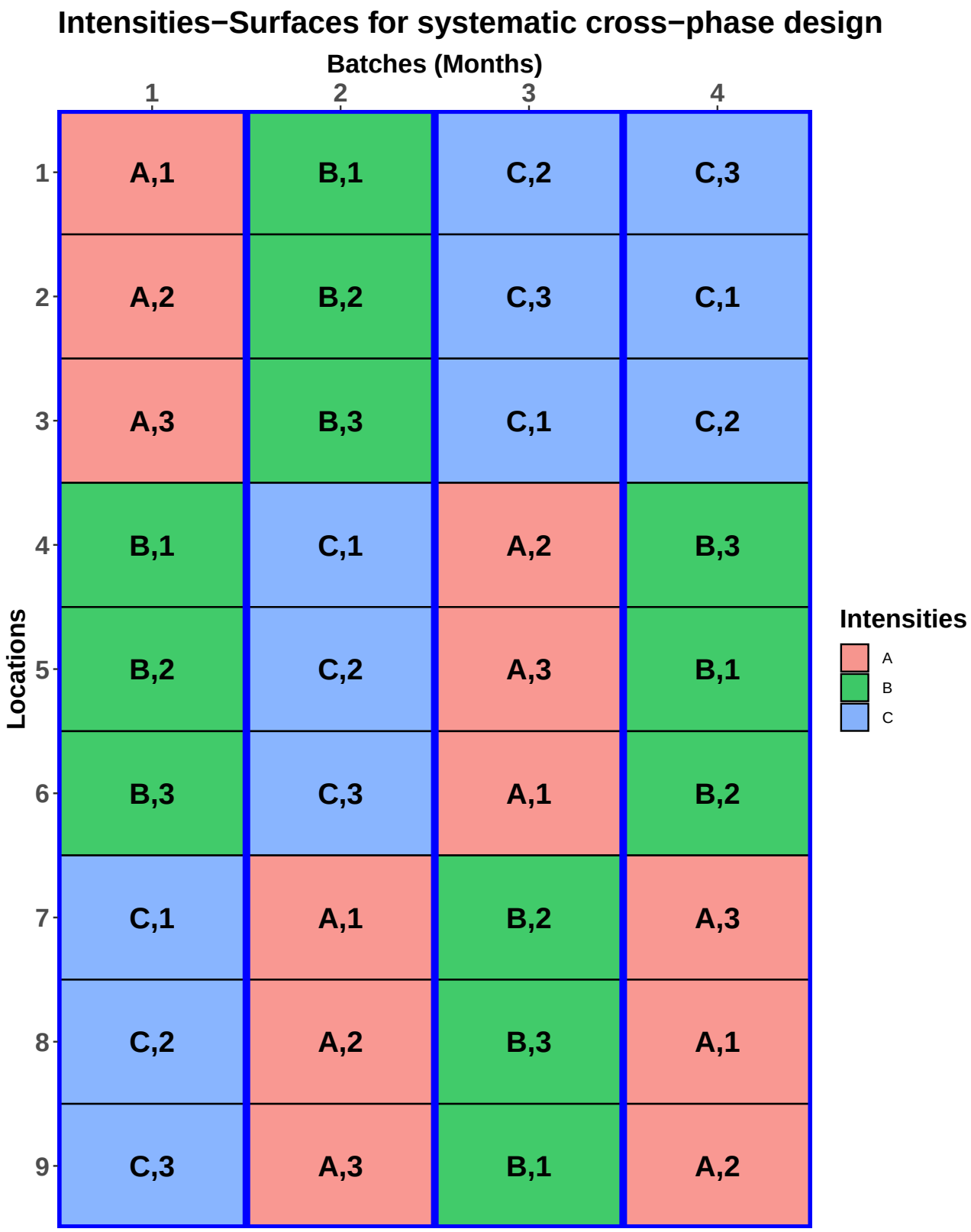


Figure 14: Factor-allocation diagram for the two-phase athlete training experiment with a row-column design on the second-phase units: training conditions are randomized to tests, then training conditions and tests are randomized to locations; the '●' indicates that the observed combinations of the levels 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.

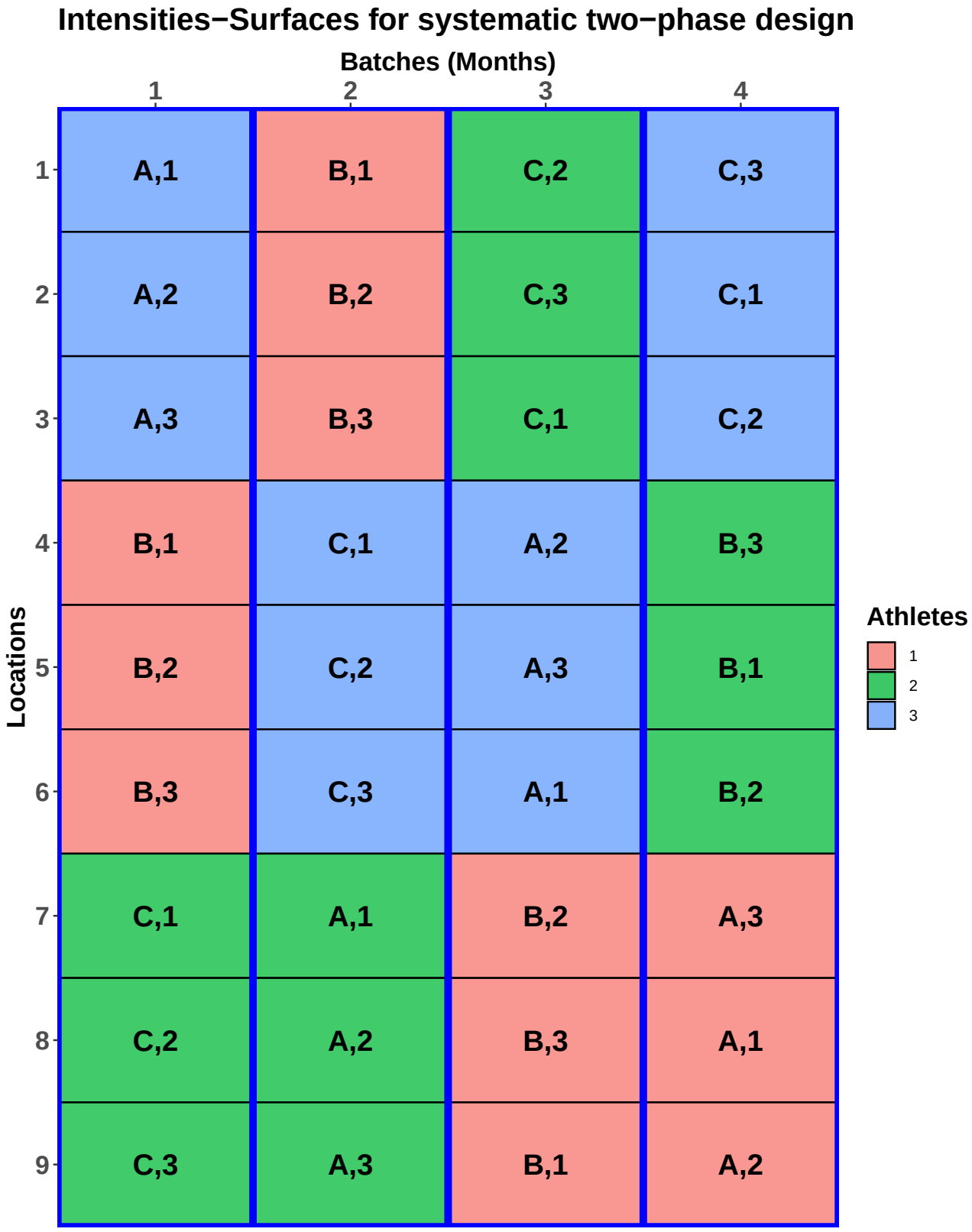
Use the following R code to obtain a layout for the new second phase design.

```
#'## Generate a systematic cross-phase design for Intensities and Surfaces
#' It is based on (i) an extended Latin square design (ELSqD) for allocating Intensities to
#' Locations triples x Batches and (ii) the same ELSqD for each triple, the ELSqD being used to
#' allocate Surfaces to the three Locations within each triple by four Batches.
#' The Batches to which the repeated columns of the ELSqD for Intensities are assigned must be
#' different from the Batches to which repeated columns of the ELSqD 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")))
```

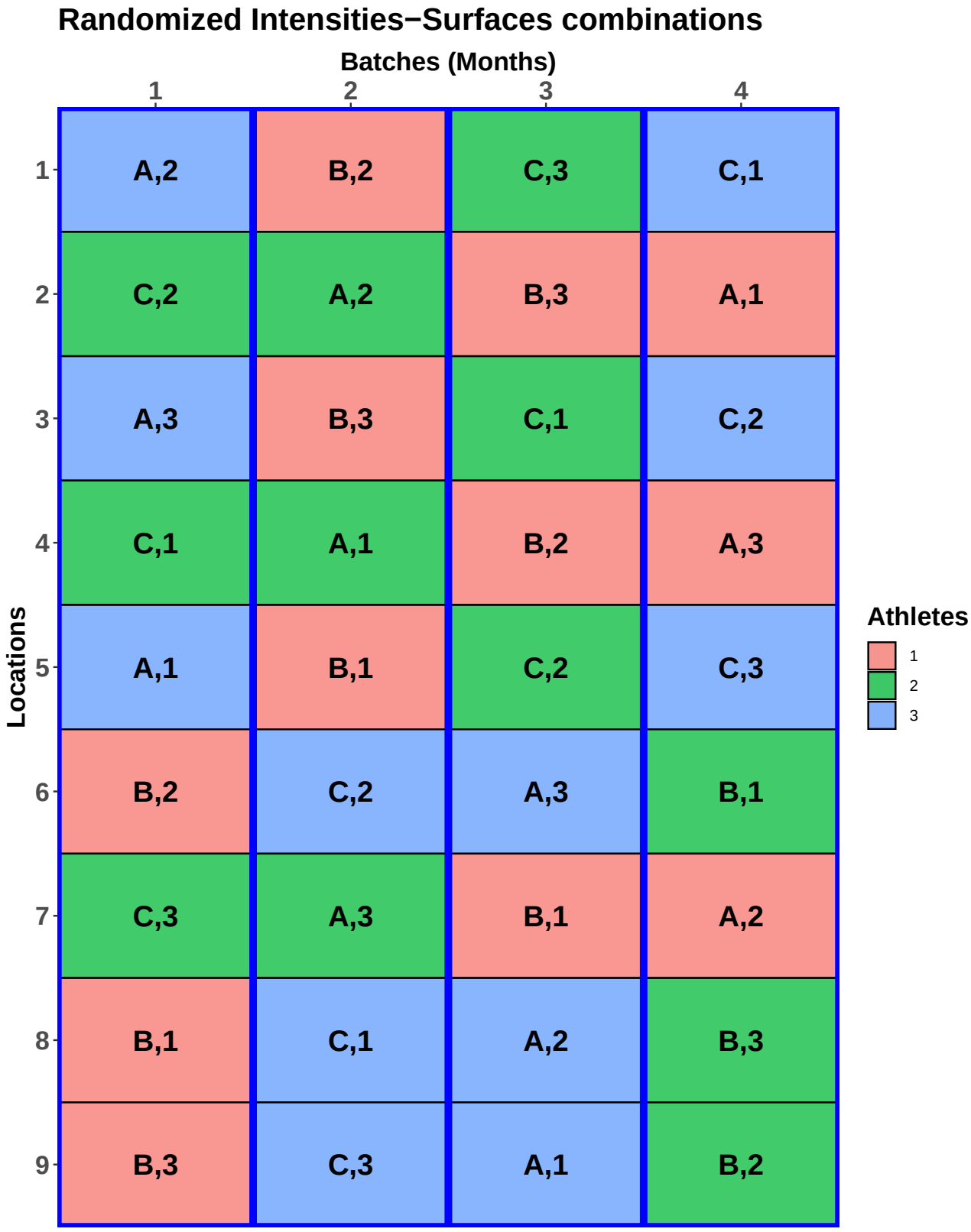


```
#'## Two-phase design
#'## Generate a systematic two-phase design by bringing in first-phase recipient factors
eg2.phx.sys$Months <- eg2.phx.sys$Batches
eg2.sys <- merge(split.lay, eg2.phx.sys) #merge on common factors Months, Intensities & Surfaces
eg2.sys  <- with(eg2.sys, eg2.sys[order(Batches,Locations),])
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 to the second-phase units
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 anatomy 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.

4.2 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.

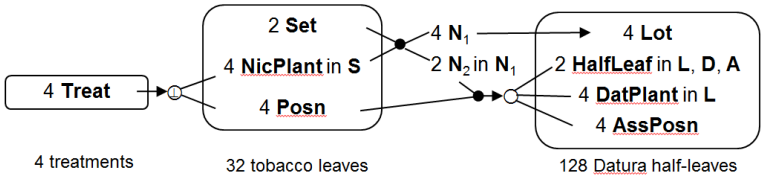


Figure 15: Factor-allocation diagram for McIntyre's (1955) two-phase experiment: treatments are randomized to tobacco leaves and tobacco leaves are randomized to Datura half-leaves; the arrow to the '⊥', the '⊥' and the two genotypes from the '⊥' indicate that Treat is randomized to the combinations 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
1	a	b	c	d
	1	5	9	13
2	b	a	d	c
	2	6	10	14
3	c	d	a	b
	3	7	11	15
4	d	c	b	a
	4	8	12	16

Leaf Position	1	2	3	4
	a	b	c	d
	17	21	25	29
	c	d	a	b
	18	22	26	30
	d	c	b	a
	19	23	27	31
	b	a	d	c
	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
1	1	2	3	4
	17	20	18	19
2	2	1	4	3
	18	19	17	20
3	3	4	1	2
	19	18	20	17
4	4	3	2	1
	20	17	19	18

Assay Leaf Position	5	6	7	8
	5	6	7	8
	23	22	24	21
	8	7	6	5
	22	23	21	24
	7	8	5	6
	21	24	22	23
	6	5	8	7
	24	21	23	22

Datura Plants

Assay Leaf Position	9	10	11	12
1	9	10	11	12
	28	25	27	26
2	10	9	12	11
	27	26	28	25
3	11	12	9	10
	26	27	25	28
4	12	11	10	9
	25	28	26	27

Assay Leaf Position	13	14	15	16
	13	14	15	16
	30	31	29	32
	16	15	14	13
	31	30	32	29
	15	16	13	14
	32	29	31	30
	14	13	16	15
	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.

4.2.1 Check the properties of the randomized layout

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

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

##
## ### Anatomy for the full two-phase design
##
##
## 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
##
##
## ### Anatomy for the first-phase design
##
##
## Summary table of the decomposition for test & trt
##
##  Source.test        df1 Source.trt df2 aefficiency order
##  Set                  1
##  NicPlant[Set]        6
##  Posn                 3
##  Set#Posn             3
##  NicPlant#Posn[Set]  18 Treat        3      1.0000     1
##                         Residual    15

## Warning in print.summary.pcanon(summary(twoph1.lay.canon, which.criteria = which.criteria)):
The combined dimensions of the sources from the first formula are less than the number of rows
in data
```

```
##
## ### Anatomy for the cross-phase, treatments design
##
##
## Summary table of the decomposition for assay & trt (based on adjusted quantities)
##
##  Source.assay                    df1 Source.trt df2 aefficiency order
##  Lot                               3
##  DatPlant[Lot]                    12
##  AssPosn                           3
##  Lot#AssPosn                       9
##  DatPlant#AssPosn[Lot]            36 Treat        3      0.5000     1
##                                      Residual    33
##  HalfLeaf[Lot:DatPlant:AssPosn]   64 Treat        3      0.5000     1
##                                      Residual    61
##
## The design is not orthogonal
##
##
## ### Anatomy for the combined-units design
##
##
## Summary table of the decomposition for assay & test (based on adjusted quantities)
##
##  Source.assay                    df1 Source.test        df2 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      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      0.5000     1
##                                      Residual            36
##
## The design is not orthogonal
```

4.2.2 Questions

1. Summarize the properties of the four design species for this example.

   The first phase design is orthogonal. However, the other three designs are nonorthogonal, but balanced. Clearly, the lack of orthogonality is introduced in the second phase.

2. 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].

3. 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.

4.3 A p-rep design for a field experiment with 576 Genotypes

A field experiment is to be conducted on a grid of 60 rows × 12 columns. Of the 576 Genotypes, 144 are to be duplicated and the remaining 432 are to be unreplicated. In the lecture, the field-phase design was optimized under a model in which Genotypes was assumed to be random. However, in the resulting design, all border plots were occupied by duplicated lines. As Sermarini et al. (2020) note, this can be avoided by assuming Genotypes to be fixed with little loss in precision. Here we use `odw` to obtain a near-optimal design under a model with fixed Genotypes. The factor-allocation diagram is in Figure 18.

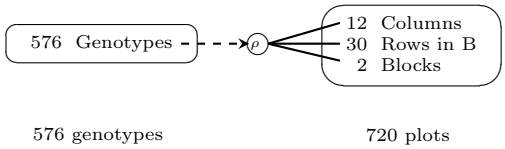


Figure 18: Factor-allocation diagram for the p-rep design for a field experiment with 576 Genotypes: genotypes are allocated to plots; the dashed arrow on the left indicates that the allocation of Genotypes is not randomized; the '(ρ)' at the end of the arrow indicates that Genotypes are allocated to combinations of the levels of Blocks, Rows and Columns, using a design that takes into account correlation between plots; B = Blocks.

4.3.1 Generate the starting design and check the properties of the design

Use the following R code to generate a balanced-lattice design and to check its properties.

```
#'# This script generates a p-rep design for 576 genotypes, 144 of which are replicated and Genotypes a
#'### It is the first-phase design of a two-phase design a la Smith et al. (2006)

#'## Set up constants
g <- 576    # no. genotypes
ndup <- 144 # no. duplicated genotypes
b <- 2      # no. blocks
r <- 60     # no. rows
c <- 12     # no. columns
n <- r*c    # no. plots

#'## Generate a simple lattice for Genotypes 1:144
#'
#' 1:144 are replicated twice 145:g are replicated once
latt.mat <- matrix(1:ndup, nrow = 12, ncol = 12)
blk1.genos <- sample((ndup+1):g, (g-ndup)/2)     #randomly select half undup Genotypes for Block 1
blk2.genos <- ((ndup+1):g)[!((ndup+1):g %in% blk1.genos)] #rest in Block 2
latt.lay <- fac.gen(list(Blocks = 2, WRows = 30, Columns = 12))
latt.lay <- within(latt.lay,
                   Genotypes <- factor(c(latt.mat, blk1.genos,
                                         t(latt.mat),blk2.genos)))
```

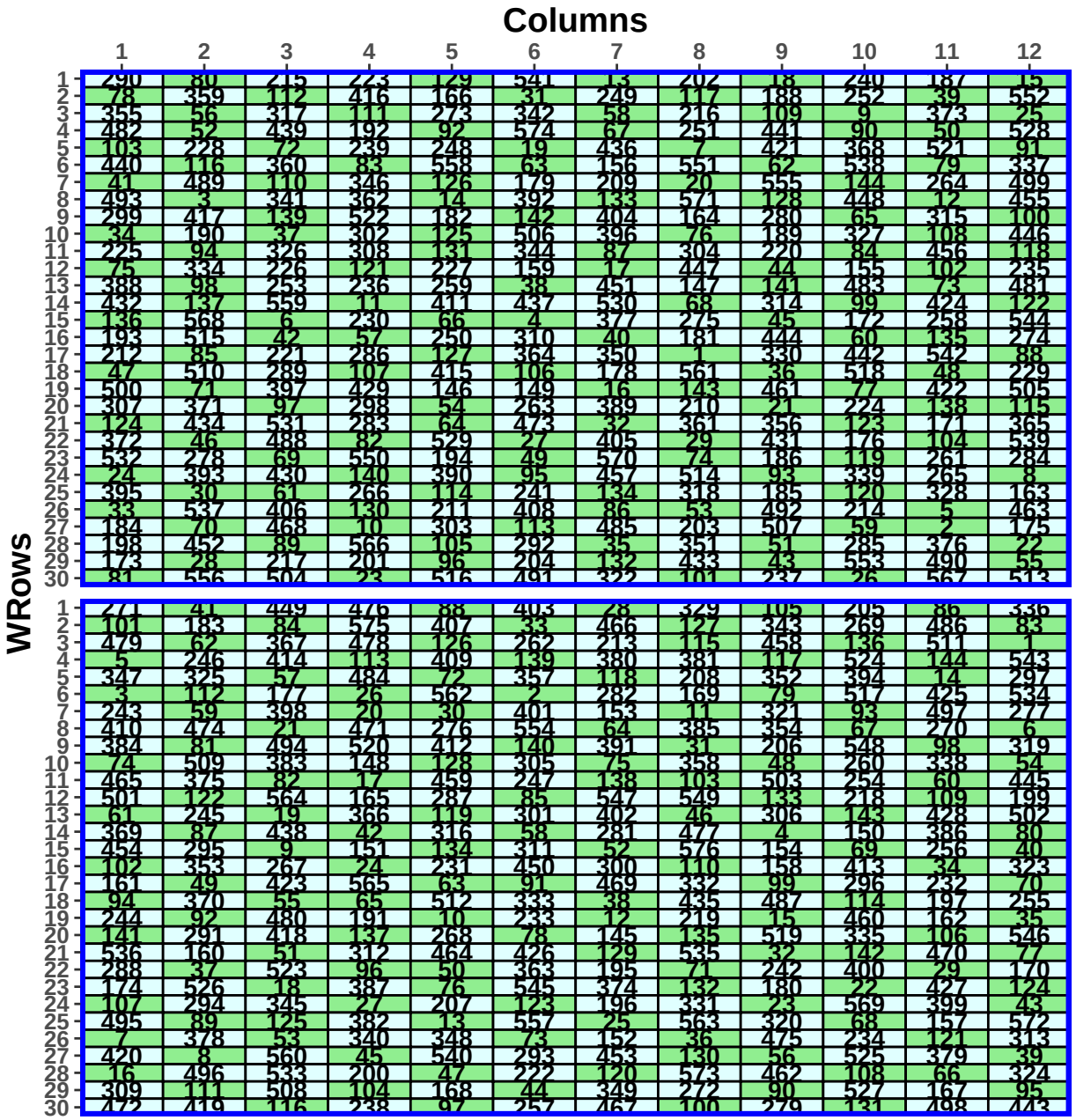
```
#'### Randomize the initial design
latt.lay <- designRandomize(allocated         = latt.lay["Genotypes"],
                            recipient         = latt.lay[c("Blocks", "WRows", "Columns")],
                            nested.recipients = list(WRows = "Blocks"),
                            seed              = 64058)
latt.lay <- within(latt.lay,
                   Rows <- fac.combine(list(Blocks, WRows)))

#'### Check properties
latt.canon <- designAnatomy(formulae = list(plot = ~ (Blocks + Rows)*Columns,
                                            trt  = ~ Genotypes),
                            data     = latt.lay)
summary(latt.canon, which.criteria = c("aeff", "meff", "eeff", "order", "dfor"))

##
##
## Summary table of the decomposition for plot & trt (based on adjusted quantities)
##
##  Source.plot          df1 Source.trt df2 aefficiency mefficiency eefficiency order dforthog
##  Blocks                 1 Genotypes    1      0.6000      0.6000      0.6000     1        0
##  Rows[Blocks]          58 Genotypes   58      0.7117      0.8000      0.4000     3       35
##  Columns               11 Genotypes   11      0.8000      0.8000      0.8000     1        0
##  Blocks#Columns        11 Genotypes   11      0.8000      0.8000      0.8000     1        0
##  Rows#Columns[Blocks] 638 Genotypes  517      0.9821      0.9872      0.7000     2      495
##                           Residual   121
##
## The design is not orthogonal
```

4.3.2 Search for a near-A-optimal design

Use `odw` to search for a near-A-optimal design under a mixed model with Genotypes assumed fixed.

```
#'## Set odw options
maxit <- 10
search <- "tabu+rw"
odw.options(P = 0.10, localSearch = 10000, tabuStop = 100)

#'## Set up variance parameters (based on Smith et al (2006, p.405))
g.G   <- 1
g.BR  <- 0.5
g.C   <- 0.1
g.BC  <- 0.05
g.u   <- 0.5
g.BRC <- 1.0
rho.R <- 0.6
rho.C <- 0.4

prepuar1.latt.odw<- odw(fixed        = ~ Genotypes + Blocks,
                        random       = ~ Rows + Columns/Blocks + units,
                        residual     = ~ ar1(Rows):ar1(Columns),
                        permute      = ~ Genotypes, swap = ~ Blocks,
                        start.values = TRUE,
                        data         = latt.lay)
```

```
vp.table <- prepuar1.latt.odw$vparameters.table
vp.table$Value[c(1:4, 6:7)] <- c(g.BR, g.C, g.BC, g.u, rho.R, rho.C)
vp.table

##                   Component Value
## 1                      Rows  0.50
## 2                   Columns  0.10
## 3           Columns:Blocks  0.05
## 4                     units  0.50
## 5           Rows:Columns!R  1.00
## 6    Rows:Columns!Rows!cor  0.60
## 7 Rows:Columns!Columns!cor  0.40

prepuar1.latt.odw <- odw(fixed    = ~ Genotypes + Blocks,
                         random   = ~ Rows + Columns/Blocks + units,
                         residual = ~ ar1(Rows):ar1(Columns),
                         permute  = ~ Genotypes, swap = ~ Blocks,
                         G.param  = vp.table, R.param = vp.table,
                         maxit    = maxit, search = search,
                         data     = latt.lay)

## Sat Apr 08 19:13:17 2023
## Initial criterion = 3.422294 (576 A-equations; rank C 575)
## Criterion after 1000 initial random iterations: 2.786026
## Criterion after tabu loop 1 is 2.741649
## Criterion after tabu loop 2 is 2.736302
## Criterion after tabu loop 3 is 2.733667
## Criterion after tabu loop 4 is 2.733033
## Criterion after tabu loop 5 is 2.732718
## Criterion after tabu loop 6 is 2.731109
## Criterion after tabu loop 7 is 2.731109
## Criterion after tabu loop 8 is 2.730840
## Criterion after tabu loop 9 is 2.730840
## Criterion after tabu loop 10 is 2.730591
## Hash table size 1358
## Final criterion after 10 tabu+rw iterations: 2.730591
## Cleaning up: Sat Apr 08 19:14:40 2023

prepuar1.latt.lay <- prepuar1.latt.odw$design

#'### Plot the design
prepuar1.latt.lay$Replication <- fac.recast(prepuar1.latt.lay$Genotypes,
                                            newlevels = rep(1:2, c(ndup, (g-ndup))))
designGGPlot(prepuar1.latt.lay, labels = "Genotypes",
             row.factors = c("Blocks", "WRows"), column.factors = "Columns",
             cellfillcolour.column = "Replication",
             colour.values = c("lightgreen", "lightcyan"),
             axis.text.size = 10, blockdefinition = cbind(30,12),
             title = NULL)
```



Obtain the anatomy of the design produced and calculate the A-value under the model for Genotypes assumed random.

```
#'### Check properties
summary(prepuar1.latt.canon <- designAnatomy(formulae = list(plot = ~ (Blocks + Rows)*Columns,
                                                             trt  = ~ Genotypes),
                                             data     = prepuar1.latt.lay),
        which.criteria = c("aeff", "meff", "eeff", "order", "dfor"))

##
##
## Summary table of the decomposition for plot & trt (based on adjusted quantities)
##
```

```
##  Source.plot          df1 Source.trt df2 aefficiency mefficiency eefficiency order dforthog
##  Blocks                 1 Genotypes    1      0.6000      0.6000      0.6000     1        0
##  Rows[Blocks]          58 Genotypes   58      0.7904      0.8000      0.6442    58        0
##  Columns               11 Genotypes   11      0.7822      0.7833      0.7353    11        0
##  Blocks#Columns        11 Genotypes   11      0.8155      0.8167      0.7654    11        0
##  Rows#Columns[Blocks] 638 Genotypes  575      0.5095      0.8877      0.0278    82      494
##                           Residual    63
##
## The design is not orthogonal

#'### Calculate the A-measure under Genotypes random
prepuar1.latt.lay$unit <- factor(1:nrow(prepuar1.latt.lay)) #factor for ASReml units
(designAmeasures(mat.Vpredicts(target = ~ Genotypes - 1,
                               Gt     = 1,
                               fixed  = ~ Blocks,
                               random = ~ Rows + Columns/Blocks + unit - 1,
                               G      = as.list(c(g.BR, g.C, g.BC, g.u)),
                               R      = kronecker(mat.ar1(rho.R, r),
                                                  mat.ar1(rho.C, c)),
                               design = prepuar1.latt.lay)))

##          all
## all 1.017486
```

In the values for the variance parameters, γ_{BC} was set to 0.05, thus indicating that it was thought to be small. The question then arises as to what would be the effect of leaving out the term. To check this recalculate the AVPD without it and redo the anatomy with the source omitted.

```
prepuar1.latt.lay$unit <- factor(1:nrow(prepuar1.latt.lay)) #factor for ASReml units
(designAmeasures(mat.Vpredicts(target = ~ Genotypes -1,
                               Gt     = 1,
                               fixed  = ~ Blocks,
                               random = ~ Rows + Columns + unit - 1,
                               G      = as.list(c(g.BR, g.C, g.u)),
                               R      = kronecker(mat.ar1(rho.R, r),
                                                  mat.ar1(rho.C, c)),
                               design = prepuar1.latt.lay)))[[1]]

## [1] 1.015127

prepBCout.canon <- designAnatomy(formulae = list(plot = ~ (Blocks + Rows) + Columns +
                                                          Blocks:Rows:Columns,
                                                 trt  = ~ Genotypes),
                                 data     = prepuar1.latt.lay)
summary(prepBCout.canon, which.criteria = c("aeff", "meff", "eeff", "order", "dfor"))

##
##
## Summary table of the decomposition for plot & trt (based on adjusted quantities)
##
##  Source.plot          df1 Source.trt df2 aefficiency mefficiency eefficiency order dforthog
##  Blocks                 1 Genotypes    1      0.6000      0.6000      0.6000     1        0
##  Rows[Blocks]          58 Genotypes   58      0.7904      0.8000      0.6442    58        0
##  Columns               11 Genotypes   11      0.7822      0.7833      0.7353    11        0
##  Blocks#Rows#Columns 649 Genotypes  575      0.5928      0.9033      0.0396    71      505
```

```
##                           Residual    74
##
## The design is not orthogonal
```

4.3.3 Questions

1. How do the plots of the p-rep designs obtained from the balanced lattice under the assumptions of fixed and random Genotypes compare?

   It would appear the duplicated and unduplicated genotypes are better dispersed when the Genotypes are assumed fixed.

2. The A-value for the design obtained with random Genotypes was 1.202. How does A-value for the design optimized with fixed Genotypes compare when random Genotypes are assumed for the model for it?

   The A-value for the design optimized under a model with fixed Genotypes but computed under a model with random Genotypes is smaller, being about 1.017.

3. Summarize the differences between the original balanced lattice design and the `odw` design, both optimized under a fixed genotypes. Is the increased precision of the `odw` design worthwhile?

   *The AVPD has decreased from 3.422294 vs 2.732786. However, Genotypes degrees of freedom in the bottom stratum has increased from 517 to the full 575 degrees of freedom, with a corresponding decrease in the Residual df from 121 to 63. The mefficiency has decreased from 0.9872 to 0.8877, but the full efficiency of the odw design is 0.9872 * 517 / 575 = 0.8876. So the increase in precision is quite marked.*

4. Is this design connected under a fixed model? How can you tell?

   Yes, it is because all 575 df for Genotypes are at least partially confounded with the residual (or identity) term, namely Rows#Columns[Blocks].

4.4 A two-phase p/q-rep design for a field experiment with 576 Genotypes

In Section 4.3, a design was constructed for a field experiment to be conducted on a grid of 60 rows × 12 columns. Of the 576 Genotypes, 144 were duplicated and the remaining 432 were unreplicated. This field experiment is the first-phase of the experiment, the second phase being a milling phase in which samples of grain are taken from the plots to be milled so that quality characteristics of the grain can be ascertained.

The factor-allocation diagram for the two-phase experiment is in Figure 19.

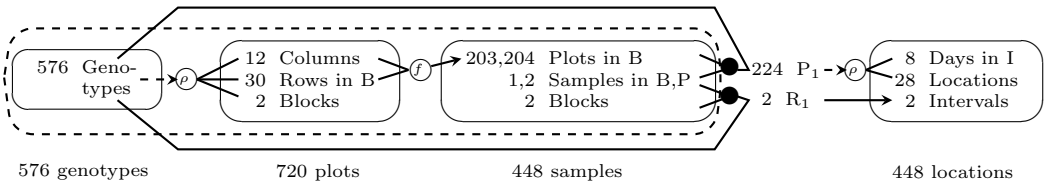
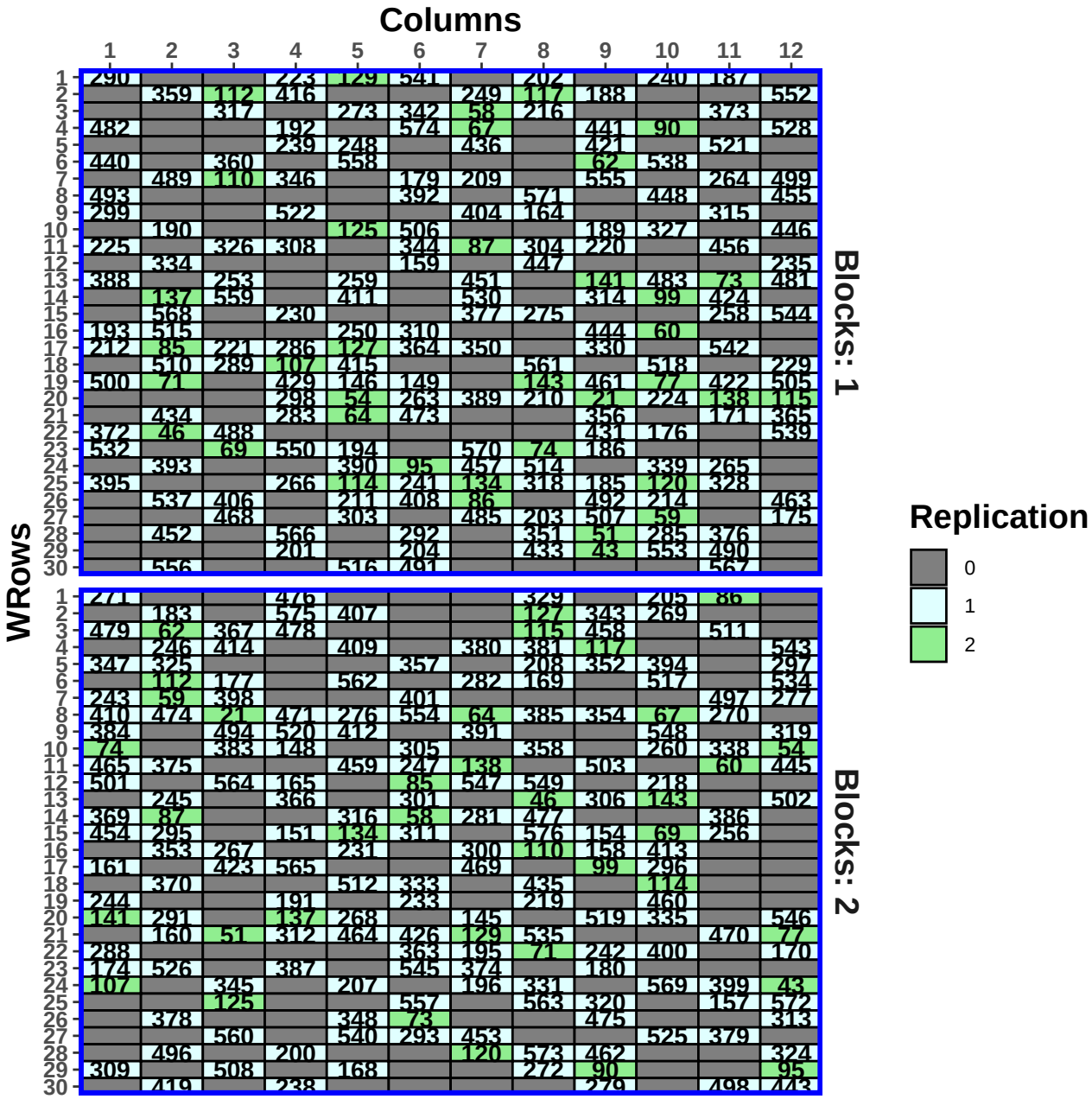


Figure 19: Factor-allocation diagram for a two-phase p/q-rep design for a field experiment with 576 Genotypes: genotypes are allocated to plots, a fraction of the plots are selected to produce samples and samples are allocated to locations; the dashed arrow on the left indicates that the allocation of Genotypes is not randomized; the 'ρ' at the end of the arrow indicates that Genotypes are allocated to combinations of the levels of Blocks, Rows and Columns, using a design that takes into account correlation between plots; the 'f' indicates the selection of a fraction of the levels of Rows and Columns from each Block; the solid lines signify that the selection is random; the dashed oval encircling the three panels on the left indicates that a pseudotier of all factors is formed in allocating samples to locations because it uses all the information about the first-phase factors; the levels of the pseudofactor R_1 groups together the blocks and samples that are to be assigned to the same interval; the pseudofactor P_1 indexes the Plots that are to occur within the same interval; the dashed arrow ending at the 'ρ' indicates that Plots within P_1 are systematically allocated, the 'ρ' indicates that the design allows for correlation between observations in the milling phase and the two lines leaving it indicate that the Plots are assigned to the combinations of the levels of Days and Locations within an Interval; B = Blocks; P = Plots; I = Intervals; D = Days.

4.4.1 Select the samples and assign them systematically to the milling phase

Use the following R code to select the samples from the field experiment for the milling phase, plot it and check its properties.

```
#'# This script systematically assigns sampled plots from the first-phase.
#'### It is based on an example from Smith et al. (2006)

#'## Select genotypes for milling phase, balanced between blocks
sampdupgenos <- sample(1:ndup, 37)               #select the 37 dup field genotypes
samp.blk1.undup <- sample(blk1.genos, 167)       #select the 167 undup field genotypes from blk1
samp.blk1.milldup <- sample(samp.blk1.undup, 20) #and from them select 20 genos to dup in milling phase
samp.blk2.undup <- sample(blk2.genos, 166)       #select the 166 undup field genotypes from blk1
samp.blk2.milldup <- sample(samp.blk2.undup, 21) #and from them Select 21 genos to dup in milling phase
milldup <- c(samp.blk1.milldup, samp.blk2.milldup)
millundup <- setdiff(c(samp.blk1.undup, samp.blk2.undup), milldup)
sampgenos <- c(sampdupgenos, millundup, milldup)

#'## Construct revised data.frame
ph2samp.lay <- with(prepuar1.latt.lay, prepuar1.latt.lay[Genotypes %in% sampgenos, ])

#'### Plot the sampled plots
#+ "SamplesGfix_v9"
fullgrid <-  merge(fac.gen(list(Blocks = 2, WRows = 30, Columns = 12)),
                   ph2samp.lay, all.x = TRUE)
fullgrid$Replication <- 1
fullgrid$Replication[as.numfac(fullgrid$Genotypes) < 145 ] <- 2
fullgrid$Replication[is.na(fullgrid$Genotypes)] <- 0
fullgrid$Replication <- factor(fullgrid$Replication)
designGGPlot(fullgrid, labels = "Genotypes",
             row.factors = c("Blocks","WRows"), column.factors = "Columns",
             cellfillcolour.column = "Replication",
             colour.values = c("grey50","lightcyan","lightgreen"),
             axis.text.size = 10, blockdefinition = cbind(30,12),
             title = NULL,
             ggplotFuncs = list(theme(legend.position = "right")))

## Warning:  Removed 313 rows containing missing values (‘geom_text()‘).
```

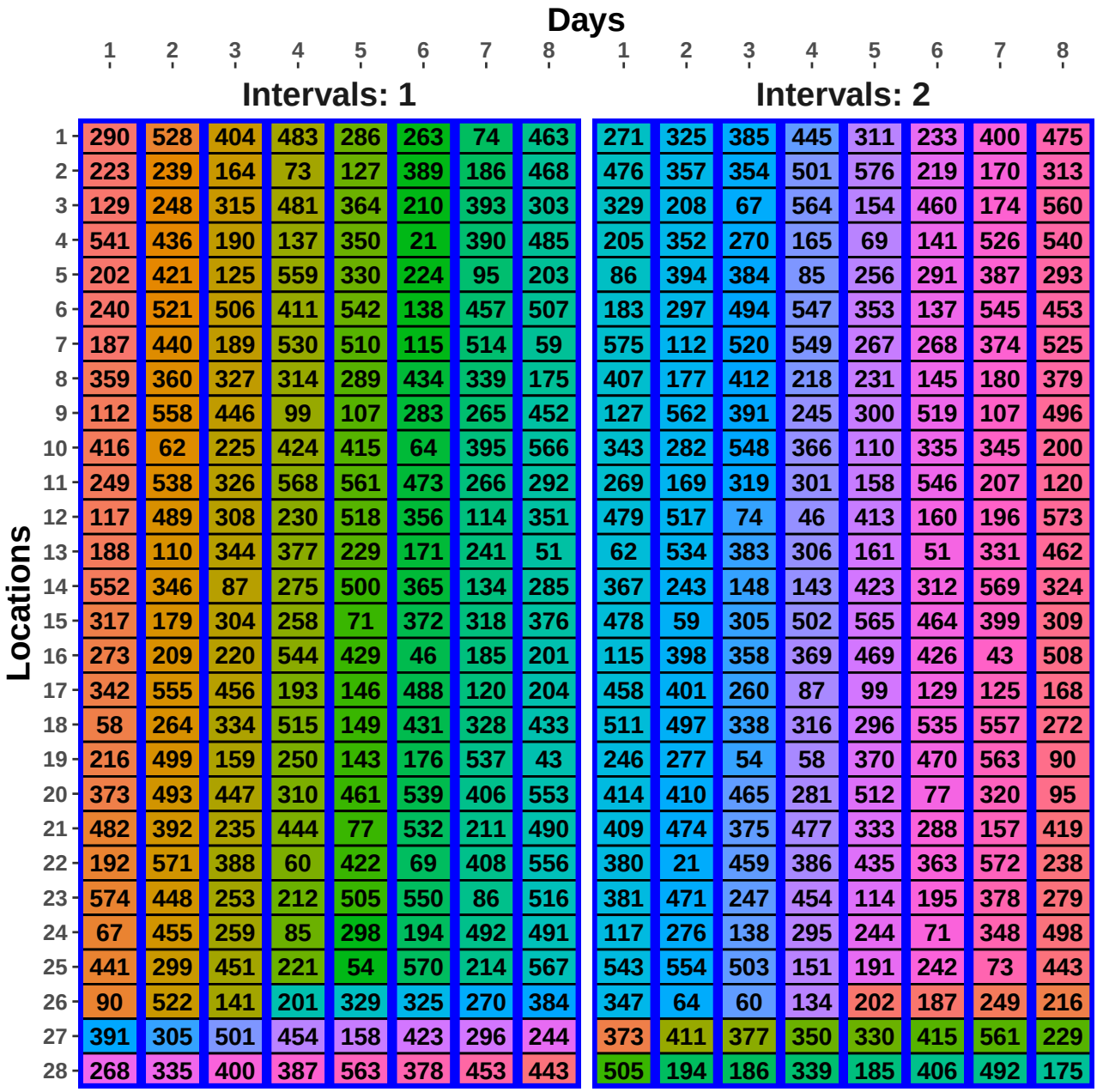



```
with(fullgrid, table(Replication, Blocks))
```

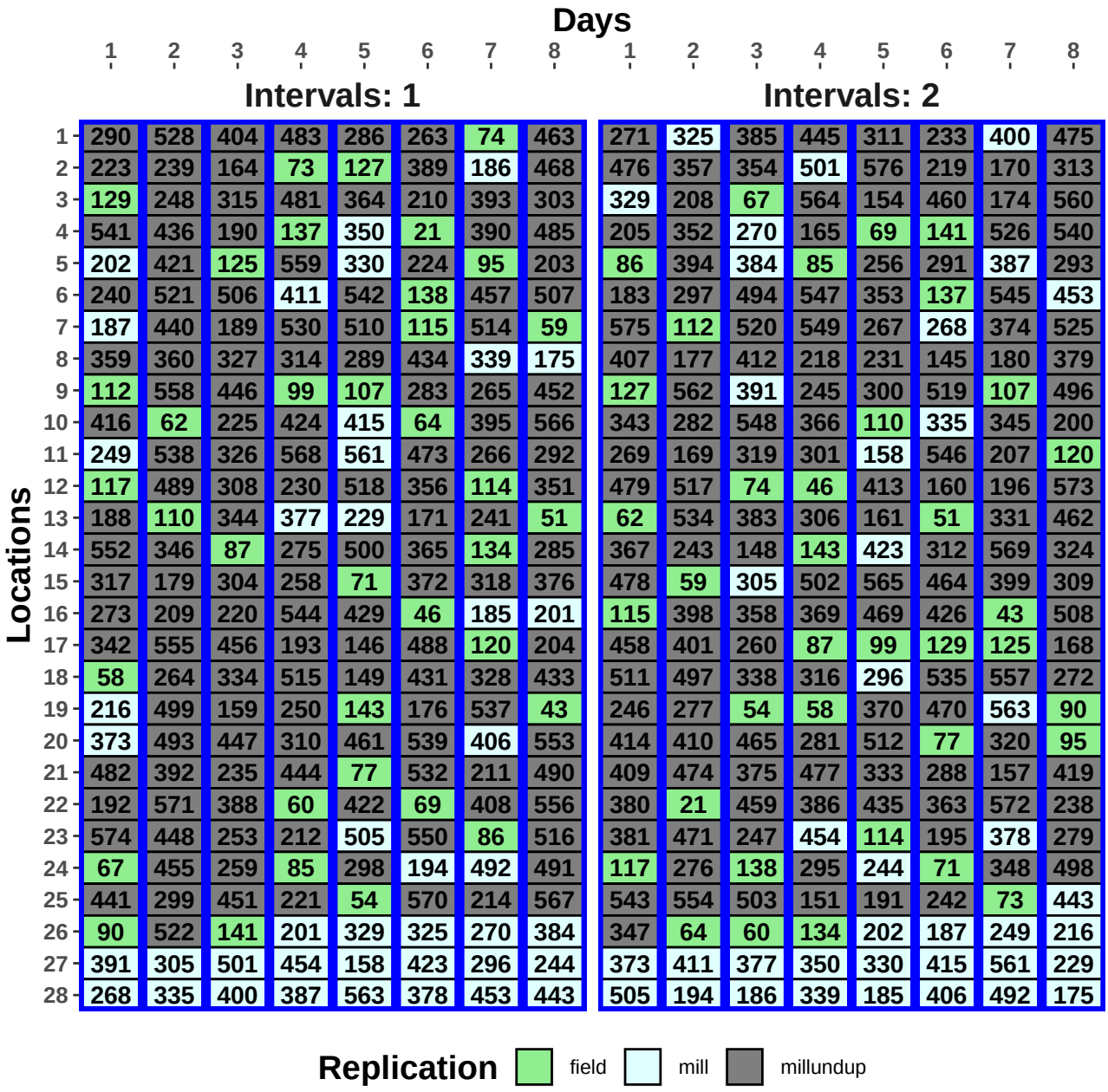
```
##             Blocks
## Replication   1   2
##           0 157 156
##           1 166 167
##           2  37  37
```

```
#'## Allocate samples systematically - confounds both field and milling dups with Intervals
#'### Allocate the field plots as the first sample
ph2sys.lay <- within(ph2samp.lay,
                     {
                       Days <- factor(c(rep(1:3, each = 26), rep(4:8, each =25),
```

```
                                        rep(1:4, each = 26), rep(5:8, each =25)))
                      Intervals <- Blocks
                      Samples <- factor(1, levels = 1:2)
                    })

#'### Add the milling duplicates in the opposite Interval to the first duplicate
ph2sys.lay <- rbind(ph2sys.lay,
                    within(prepuar1.latt.lay[prepuar1.latt.lay$Genotypes %in% milldup, ],
                           {
                             Days <- factor(c(rep(1:8, 3)[-(1:4)], rep(1:8, 3)[-(1:3)]))
                             Intervals <- Blocks
                             Intervals[Blocks == 1][1:20] <- 2
                             Intervals[Blocks == 2] <- 1
                             Samples <- factor(2, levels = 1:2)
                           }))
ph2sys.lay <- within(ph2sys.lay,
                     {
                       Genotypes <- factor(Genotypes)
                       Locations <- fac.nested(fac.combine(list(Intervals,Days)))
                       xLocn <- as.numeric(Locations)
                       xLocn <- xLocn - mean(unique(xLocn))
                     })
ph2sys.lay <- with(ph2sys.lay, ph2sys.lay[order(Intervals, Days, Locations), ])

#'### Plot the design
#+ Breed576sys2phGfix_v9
designGGPlot(ph2sys.lay, labels = "Genotypes",
             row.factors = c("Locations"), column.factors = c("Intervals", "Days"),
             cellfillcolour.column = "Rows",
             axis.text.size = 10, blockdefinition = cbind(28,1),
             title = NULL)
```



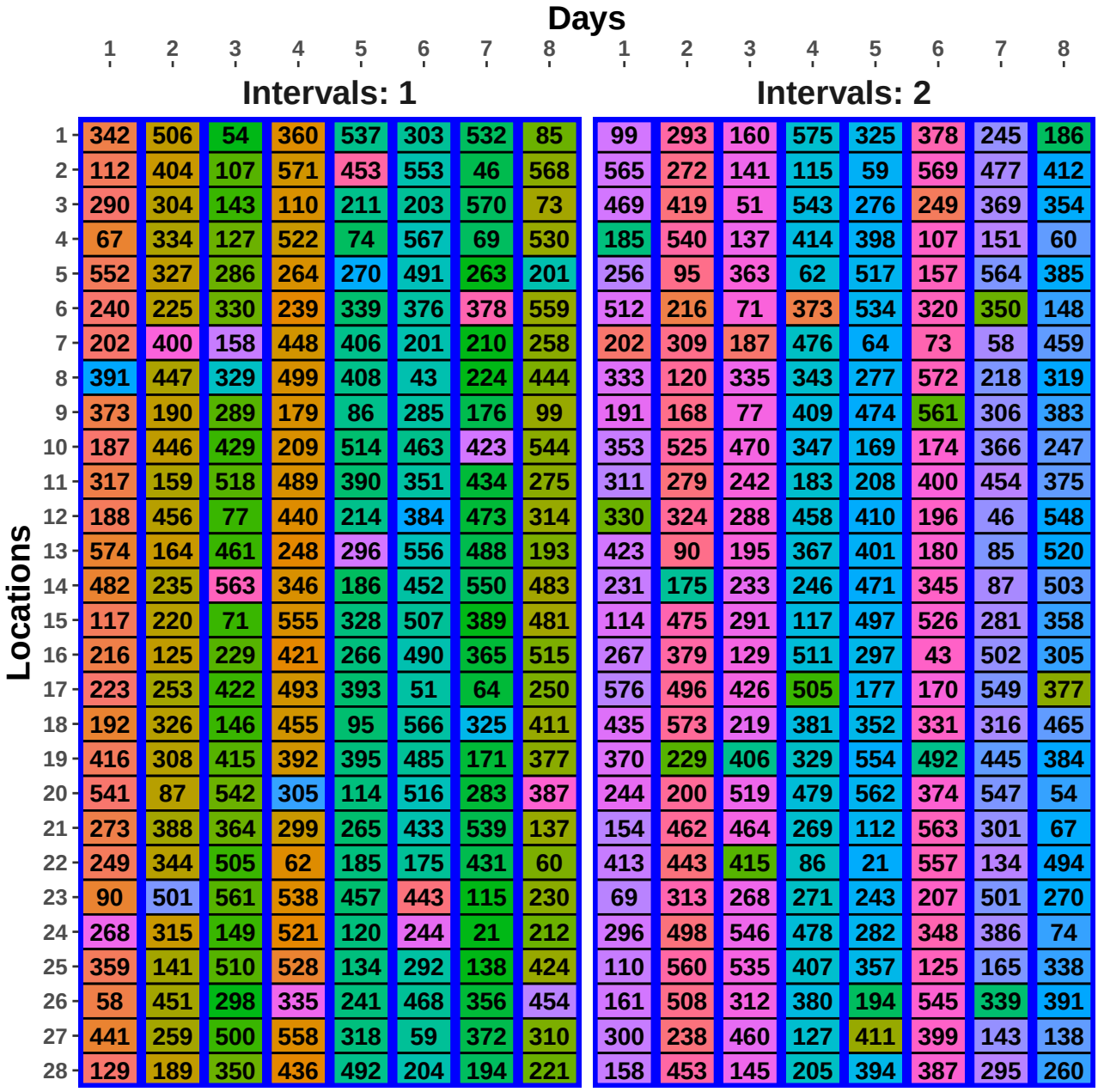

```
ph2sys.lay$Replication <- "millundup"
ph2sys.lay$Replication[ph2sys.lay$Genotypes %in% sampdupgenos] <- "field"
ph2sys.lay$Replication[ph2sys.lay$Genotypes %in% milldup] <- "mill"
designGGPlot(ph2sys.lay, labels = "Genotypes",
             row.factors = c("Locations"), column.factors = c("Intervals", "Days"),
             cellfillcolour.column = "Replication",
             colour.values = c("lightgreen","lightcyan","grey50"),
             axis.text.size = 10, blockdefinition = cbind(28,1),
             title = NULL,
             ggplotFuncs = list(theme(legend.position = "bottom")))
```



4.4.2 Randomize the systematic p/q-rep design to produce an initial design

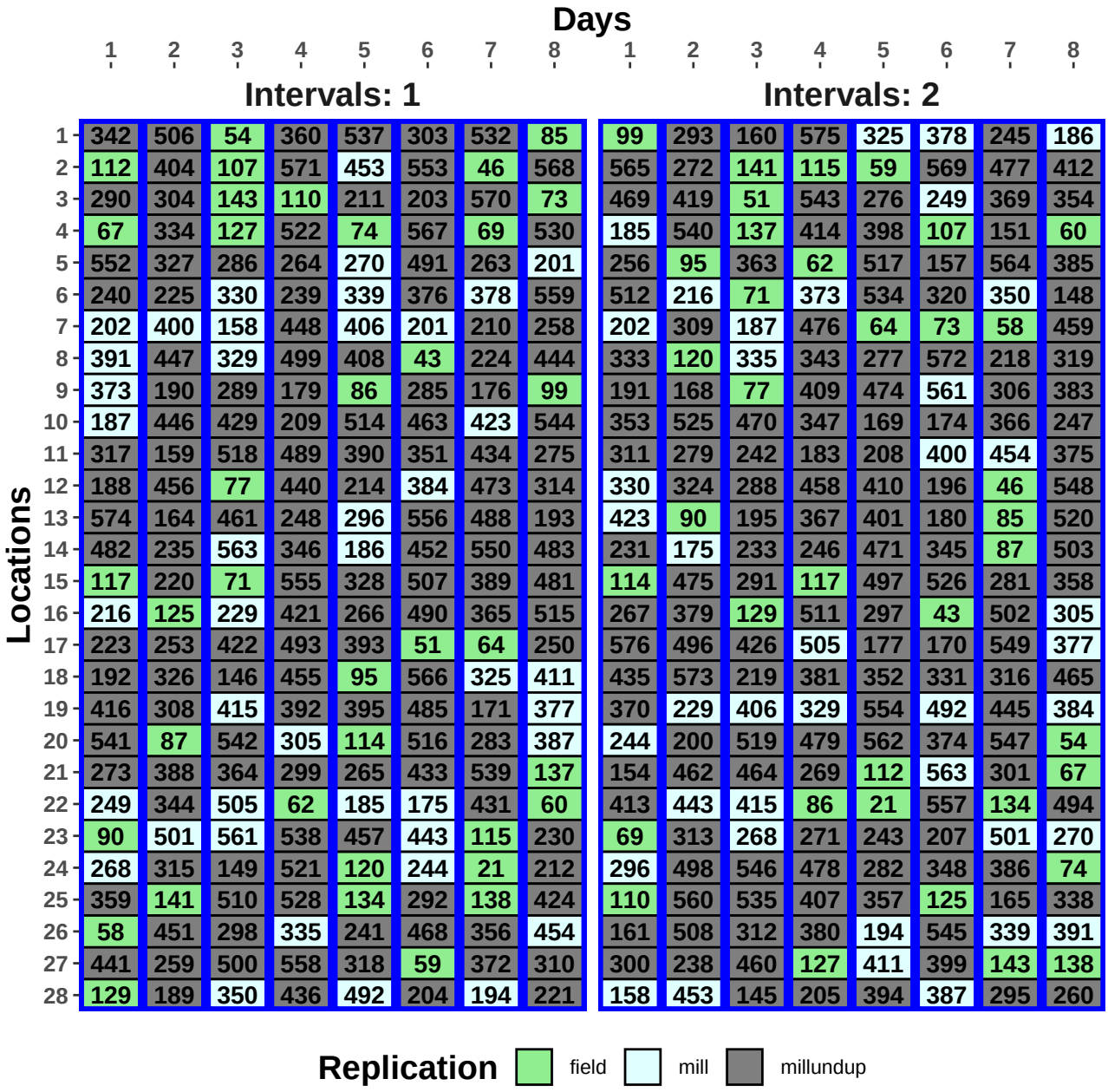
```
#'## Randomize the allocation to the second-phase units
ph2sys.lay <- with(ph2sys.lay, ph2sys.lay[order(Intervals,Days,Locations),])
ph2sys.ini <- designRandomize(allocated = ph2sys.lay[c("Blocks", "WRows", "Rows", "Columns",
                                                       "Samples", "Genotypes")],
                              recipient = ph2sys.lay[c("Intervals", "Days", "Locations")],
                              nested.recipients = list(Days = "Intervals",
                                                       Locations = c("Intervals", "Days")),
                              seed = 705865)
```

```
#'### Plot the design
designGGPlot(ph2sys.ini, labels = "Genotypes",
             row.factors = c("Locations"), column.factors = c("Intervals", "Days"),
             cellfillcolour.column = "Rows",
             axis.text.size = 10, blockdefinition = cbind(28,1),
             title = NULL)
```




```
ph2sys.ini$Replication <- "millundup"
ph2sys.ini$Replication[ph2sys.ini$Genotypes %in% sampdupgenos] <- "field"
ph2sys.ini$Replication[ph2sys.ini$Genotypes %in% milldup] <- "mill"
designGGPlot(ph2sys.ini, labels = "Genotypes",
             row.factors = c("Locations"), column.factors = c("Intervals", "Days"),
```

```
            cellfillcolour.column = "Replication",
            colour.values = c("lightgreen","lightcyan","grey50"),
            axis.text.size = 10, blockdefinition = cbind(28,1),
            title = NULL,
            ggplotFuncs = list(theme(legend.position = "bottom")))
```



4.4.3 Optimize the intial design for the two-phase model

Had to remove Rows:Columns:Samples because of singularities and Int.Days must be used in the residual model.

```
g.BRCS <- 0.25
g.ID <- 0.8
g.L <- 0.2
rho.L <- 0.6

ph2sys.ini$IntDays <- with(ph2sys.ini, fac.combine(list(Intervals, Days)))
maxit <- 3
ph2sys.odw.ini <- odw(fixed        = ~ Intervals,
                      random       = ~ Genotypes + Rows + Columns/Blocks + Rows:Columns +
                        ar1(Rows):ar1(Columns) + Rows:Columns:Samples + Intervals:Days + Locations,
                      residual     = ~ IntDays:ar1(Locations),
                      permute      = ~ Genotypes | Rows/Columns/Samples + Columns/Blocks,
                      swap         = ~ Intervals,
                      start.values = TRUE, data=ph2sys.ini)
vpc <- ph2sys.odw.ini$vparameters.table
vpc$Value <- c(g.G*g.BRC, g.BR*g.BRC, g.C*g.BRC, g.BC*g.BRC, g.BRC, rho.R, rho.C, g.BRCS,
               g.ID, g.L, 1, rho.L)
(vpc)
```

```
##                          Component Value
## 1                        Genotypes  1.00
## 2                             Rows  0.50
## 3                          Columns  0.10
## 4                   Columns:Blocks  0.05
## 5                     Rows:Columns  1.00
## 6            Rows:Columns!Rows!cor  0.60
## 7         Rows:Columns!Columns!cor  0.40
## 8             Rows:Columns:Samples  0.25
## 9                   Intervals:Days  0.80
## 10                       Locations  0.20
## 11             IntDays:Locations!R  1.00
## 12 IntDays:Locations!Locations!cor  0.60
```

```
ph2sys.odw <- odw(fixed   = ~ Intervals,
                  random  = ~ Genotypes + Rows + Columns/Blocks + Rows:Columns +
                    ar1(Rows):ar1(Columns) + Rows:Columns:Samples + Intervals:Days + Locations,
                  residual = ~ IntDays:ar1(Locations),
                  permute = ~ Genotypes | Rows/Columns/Samples + Columns/Blocks,
                  swap         = ~ Intervals,
                  G.param = vpc, R.param = vpc,
                  maxit   = maxit, search = search, data=ph2sys.ini)
```

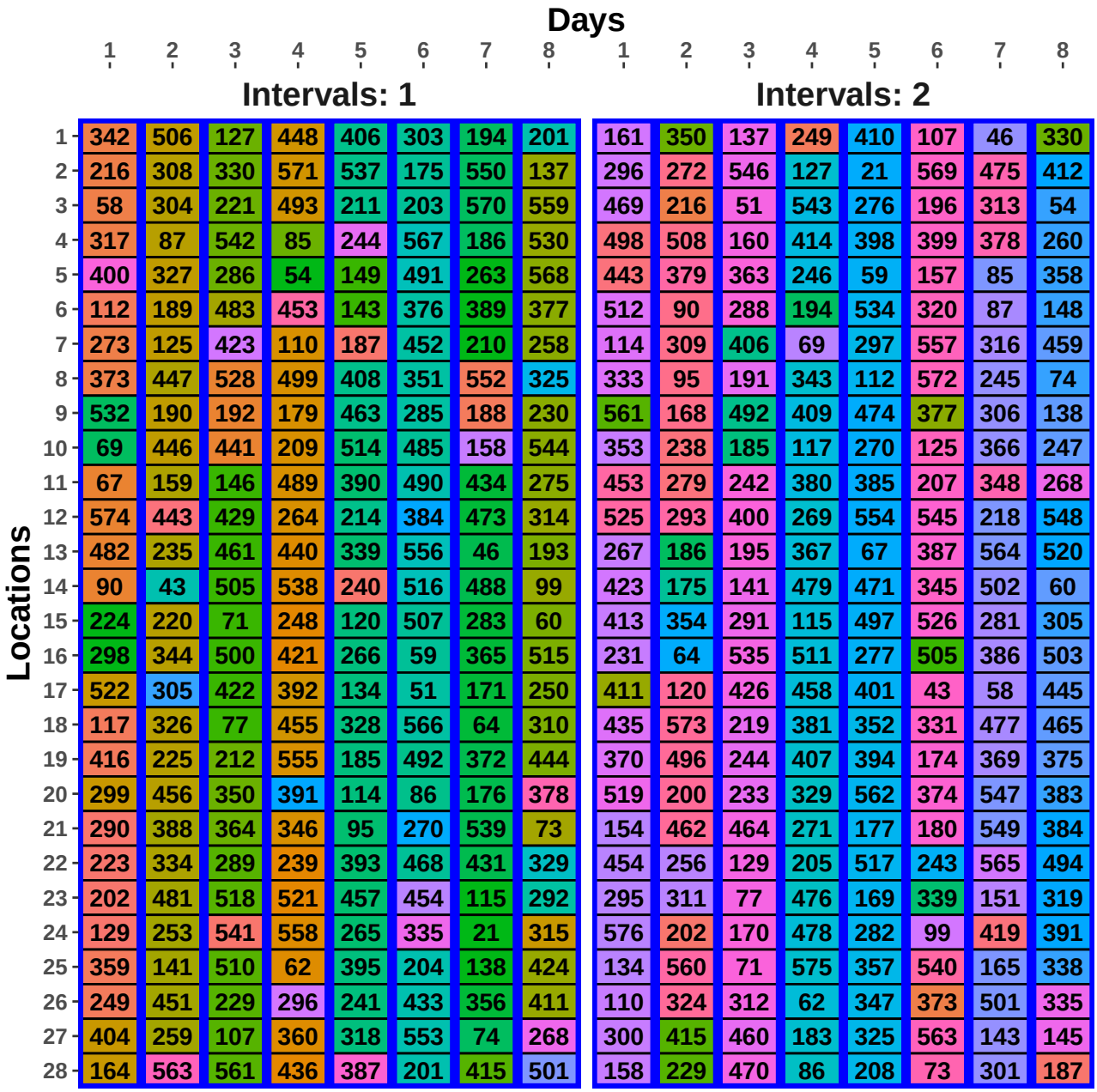
```
## Sat Apr 08 19:14:59 2023
## Moore-Penrose inverse (evd) of a 2626 X 2626 matrix - patience!
## Initial criterion = 1.493873 (370 A-equations; rank C 2626)
## Criterion after 1000 initial random iterations: 1.487506
## Criterion after tabu loop 1 is 1.484285
## Criterion after tabu loop 2 is 1.482821
## Criterion after tabu loop 3 is 1.481715
## Hash table size 384
## Final criterion after 3 tabu+rw iterations: 1.481715
## Cleaning up: Sat Apr 08 19:30:52 2023
```

```
ph2sys.odw.lay <- ph2sys.odw$design
ph2sys.odw.lay <- within(ph2sys.odw.lay,
```

```
                          {
                            WRows <- fac.recast(Rows, newlevels = rep(1:30, times = 2))
                            xLocn <- as.numeric(Locations)
                            xLocn <- xLocn - mean(unique(xLocn))
                          })

#'### Check the two-phase A-measure
G.RC <- g.BRC * kronecker(mat.ar1(rho.R, r), mat.ar1(rho.C, c)) #ar1(R):ar1(C)
(designAmeasures(mat.Vpredicts(target = ~ Genotypes - 1,
                               Gt     = g.G*g.BRC,
                               fixed  = ~ Intervals - 1,
                               random = ~ Rows + Columns/Blocks + Rows:Columns/Samples +
                                 Intervals:Days + Locations - 1,
                               G      = c(as.list(c(g.BR*g.BRC, g.C*g.BRC, g.BC*g.BRC)),
                                          list(G.RC = G.RC),
                                          as.list(c(g.BRCS, g.ID, g.L))),
                               R = with(ph2sys.odw.lay, kronecker(diag(nlevels(IntDays)),
                                                                  mat.ar1(rho.L,
                                                                          nlevels(Locations)))),
                               design = ph2sys.odw.lay)))

##          all
## all 1.305232
```

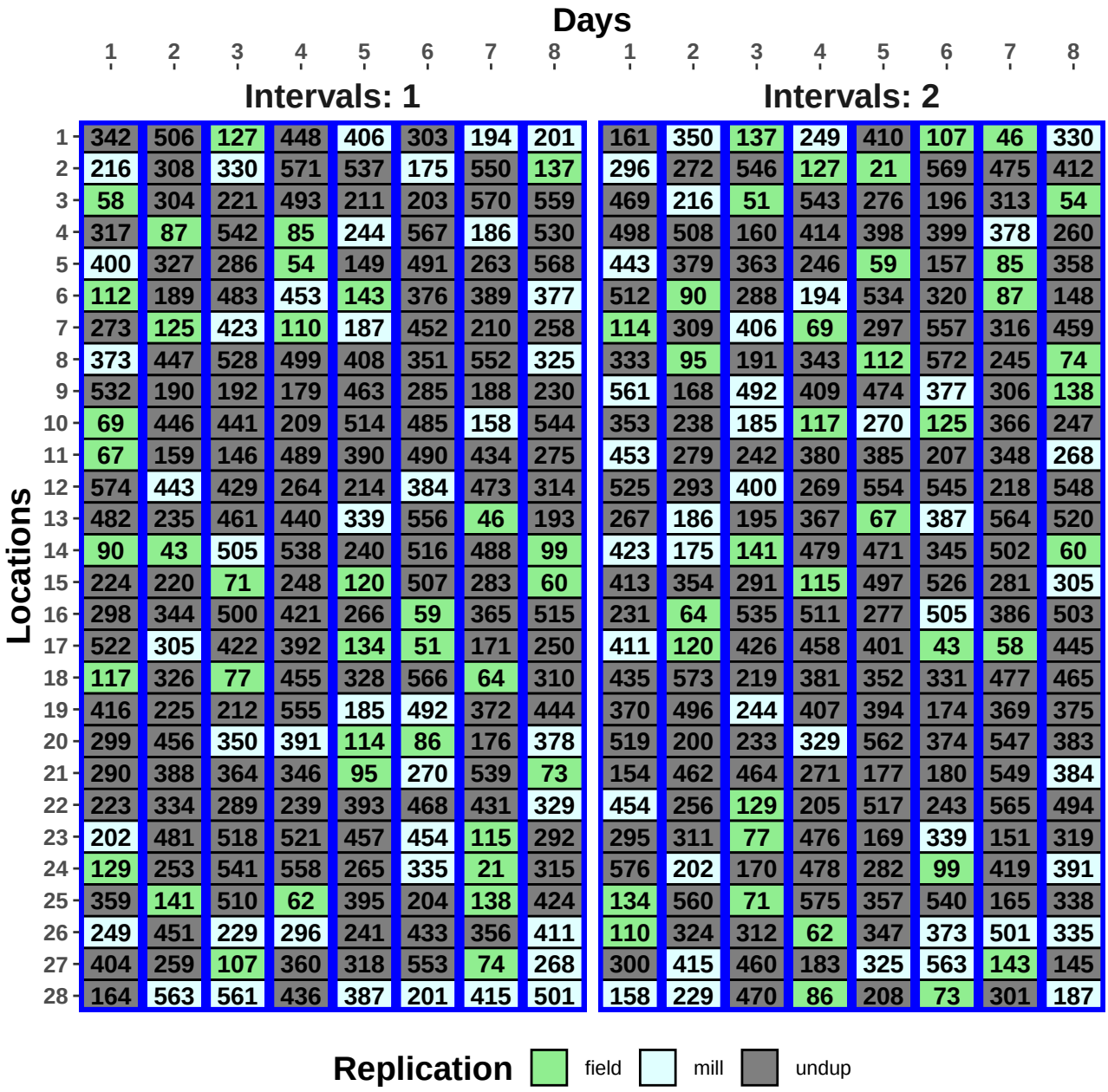
4.4.4 Plot the design

```
designGGPlot(ph2sys.odw.lay, labels = "Genotypes",
             row.factors = c("Locations"), column.factors = c("Intervals", "Days"),
             cellfillcolour.column = "Rows",
             axis.text.size = 10, blockdefinition = cbind(28,1),
             title = NULL)
```




```
#Set up a factor indicating the type of duplication
genodups <- with(ph2sys.odw.lay, table(Genotypes))
genodups <- names(genodups[genodups ==  2])
ph2sys.odw.lay$Replication <- "undup"
ph2sys.odw.lay$Replication[ph2sys.odw.lay$Genotypes %in% genodups] <- "field"
ph2sys.odw.lay$Replication[ph2sys.odw.lay$Genotypes %in% milldup] <- "mill"
designGGPlot(ph2sys.odw.lay, labels = "Genotypes",
             row.factors = c("Locations"), column.factors = c("Intervals", "Days"),
             cellfillcolour.column = "Replication",
             colour.values = c("lightgreen","lightcyan","grey50"),
             axis.text.size = 10, blockdefinition = cbind(28,1),
             title = NULL,
```

```
    ggplotFuncs = list(theme(legend.position = "bottom")))
```



4.4.5 Check the properties of the optimized p/q-rep design

```
#'### Check the replications
(with(ph2sys.odw.lay, table(Days,Replication,Intervals)))

## , , Intervals = 1
##
##     Replication
```

```
## Days field mill undup
##    1     7    5    16
##    2     4    3    21
##    3     4    6    18
##    4     4    3    21
##    5     5    6    17
##    6     3    7    18
##    7     6    4    18
##    8     4    8    16
##
## , , Intervals = 2
##
##     Replication
## Days field mill undup
##    1     3    8    17
##    2     4    7    17
##    3     6    5    17
##    4     6    3    19
##    5     4    2    22
##    6     5    6    17
##    7     5    2    21
##    8     4    7    17
```

```
(with(ph2sys.odw.lay, table(Replication,Intervals)))
```

```
##             Intervals
## Replication   1   2
##       field  37  37
##       mill   42  40
##       undup 145 147
```

```
#'### Substitute shorter factor names and produce the anatomies for all 4 design species
layout <- ph2sys.odw.lay
names(layout)[match(c("Intervals", "Locations", "Columns","Samples"), names(layout))] <-
  c("Int", "Locn", "Cols","Samp")
designTwophaseAnatomies(formulae       = list(lab = ~ (Int/Days)*Locn,
                                              plot = ~ ((Blocks/WRows)*Cols)/Samp,
                                              trt  = ~ Genotypes),
                        which.criteria = c("ae", "me", "ee", "dfor"),
                        keep.order     = TRUE, data = layout)
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows[Blocks] and Blocks
are partially aliased in Days[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows[Blocks] and Blocks
are partially aliased in Locn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows[Blocks] and Blocks
are partially aliased in Int#Locn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows[Blocks] and Blocks
are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Cols and Blocks are partially
aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Cols and WRows[Blocks]
are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Blocks#Cols and Blocks
are partially aliased in Days#Locn[Int]
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Blocks#Cols and WRows[Blocks]
are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Blocks#Cols and Cols
are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows#Cols[Blocks] and
Blocks are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows#Cols[Blocks] and
WRows[Blocks] are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows#Cols[Blocks] and
Cols are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows#Cols[Blocks] and
Blocks#Cols are partially aliased in Days#Locn[Int]

##
## ### Anatomy for the full two-phase design
##
##
## Summary table of the decomposition for lab, plot & trt (based on adjusted quantities)
##
##  Source.lab     df1 Source.plot        df2 Source.trt df3 aefficiency mefficiency eefficiency
##  Int              1 Blocks               1 Genotypes    1      0.6562      0.6562      0.6562
##  Days[Int]       14 Blocks               1 Genotypes    1      0.8644      0.8644      0.8644
##                     WRows[Blocks]       13 Genotypes   13      0.8185      0.8226      0.7231
##  Locn            27 Blocks               1 Genotypes    1      0.8186      0.8186      0.8186
##                     WRows[Blocks]       26 Genotypes   26      0.8069      0.8195      0.5822
##  Int#Locn        27 Blocks               1 Genotypes    1      0.7696      0.7696      0.7696
##                     WRows[Blocks]       26 Genotypes   26      0.8239      0.8346      0.6173
##  Days#Locn[Int] 378 Blocks               1 Genotypes    1      0.6221      0.6221      0.6221
##                     WRows[Blocks]       58 Genotypes   58      0.8263      0.8575      0.4664
##                     Cols                11 Genotypes   11      0.8846      0.8870      0.8020
##                     Blocks#Cols         11 Genotypes   11      0.8859      0.8888      0.7887
##                     WRows#Cols[Blocks] 297 Genotypes  297      0.2998      0.8158      0.0087
##  dforthog
##         0
##         0
##         0
##         0
##         0
##         0
##         0
##         0
##         0
##         0
##       219
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source             df Alias                    In             aefficiency mefficiency eefficiency
##  Cols                1 Blocks                   plot                0.0136      0.0136      0.0136
##  Cols               11 WRows[Blocks]            plot                0.0569      0.0776      0.0257
##  Cols               11 ## Information remaining plot                0.9195      0.9213      0.8491
##  Blocks#Cols        22 WRows[Blocks]            plot                0.0424      0.0778      0.0076
##  Blocks#Cols        11 ## Information remaining plot                1.0000      1.0000      1.0000
```

```
##  WRows[Blocks]       1 Blocks                  Days[Int]           1.0000      1.0000      1.0000
##  WRows[Blocks]       1 Blocks                  Locn                1.0000      1.0000      1.0000
##  WRows[Blocks]       1 Blocks                  Int#Locn            1.0000      1.0000      1.0000
##  WRows[Blocks]       1 Blocks                  Days#Locn[Int]      0.2131      0.2131      0.2131
##  Cols                1 Blocks                  Days#Locn[Int]      0.0199      0.0199      0.0199
##  Cols               11 WRows[Blocks]           Days#Locn[Int]      0.0328      0.0478      0.0128
##  Blocks#Cols         1 Blocks                  Days#Locn[Int]      0.0135      0.0135      0.0135
##  Blocks#Cols        11 WRows[Blocks]           Days#Locn[Int]      0.0329      0.0521      0.0123
##  Blocks#Cols        11 Cols                    Days#Locn[Int]      0.0001      0.0064      0.0000
##  WRows#Cols[Blocks]  1 Blocks                  Days#Locn[Int]      0.5986      0.5986      0.5986
##  WRows#Cols[Blocks] 58 WRows[Blocks]           Days#Locn[Int]      0.0727      0.5100      0.0056
##  WRows#Cols[Blocks] 11 Cols                    Days#Locn[Int]      0.4438      0.5111      0.2140
##  WRows#Cols[Blocks] 11 Blocks#Cols             Days#Locn[Int]      0.4566      0.5128      0.2721
##  dforthog
##         0
##         0
##         0
##         0
##        11
##         1
##         1
##         1
##         0
##         0
##         0
##         0
##         0
##         0
##         0
##         5
##         0
##         0
##
## The design is not orthogonal
##
##
## ### Anatomy for the first-phase design
##
##
## Summary table of the decomposition for plot & trt (based on adjusted quantities)
##
##  Source.plot             df1 Source.trt df2 aefficiency mefficiency eefficiency dforthog
##  Blocks                    1 Genotypes    1      0.8348      0.8348      0.8348        0
##  WRows[Blocks]            58 Genotypes   58      0.9135      0.9220      0.7056       21
##  Cols                     11 Genotypes   11      0.9076      0.9099      0.8384        0
##  Blocks#Cols              11 Genotypes   11      0.9170      0.9190      0.8392        0
##  WRows#Cols[Blocks]      325 Genotypes  325      0.5403      0.9064      0.0315      288
##  Samp[Blocks:WRows:Cols]  41
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source      df Alias                   In   aefficiency mefficiency eefficiency dforthog
##  Cols         1 Blocks                  plot      0.0136      0.0136      0.0136        0
```

```
##  Cols        11 WRows[Blocks]            plot      0.0569      0.0776      0.0257        0
##  Cols        11 ## Information remaining plot      0.9195      0.9213      0.8491        0
##  Blocks#Cols 22 WRows[Blocks]            plot      0.0424      0.0778      0.0076        0
##  Blocks#Cols 11 ## Information remaining plot      1.0000      1.0000      1.0000       11
##
## The design is not orthogonal
##
##
## ### Anatomy for the cross-phase, treatments design
##
##
## Summary table of the decomposition for lab & trt (based on adjusted quantities)
##
##  Source.lab     df1 Source.trt df2 aefficiency mefficiency eefficiency dforthog
##  Int              1 Genotypes    1      0.6562      0.6562      0.6562        0
##  Days[Int]       14 Genotypes   14      0.8213      0.8256      0.7222        0
##  Locn            27 Genotypes   27      0.8052      0.8194      0.5780        0
##  Int#Locn        27 Genotypes   27      0.8197      0.8322      0.5990        0
##  Days#Locn[Int] 378 Genotypes  366      0.1477      0.8530      0.0015      300
##                     Residual    12
##
## The design is not orthogonal
```

```
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows[Blocks] and Blocks
are partially aliased in Days[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows[Blocks] and Blocks
are partially aliased in Locn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows[Blocks] and Blocks
are partially aliased in Int#Locn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows[Blocks] and Blocks
are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Cols and Blocks are partially
aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Cols and WRows[Blocks]
are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Blocks#Cols and Blocks
are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Blocks#Cols and WRows[Blocks]
are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Blocks#Cols and Cols
are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows#Cols[Blocks] and
Blocks are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows#Cols[Blocks] and
WRows[Blocks] are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows#Cols[Blocks] and
Cols are partially aliased in Days#Locn[Int]
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): WRows#Cols[Blocks] and
Blocks#Cols are partially aliased in Days#Locn[Int]
```

```
##
## ### Anatomy for the combined-units design
##
##
```

```
## Summary table of the decomposition for lab & plot (based on adjusted quantities)
##
##  Source.lab     df1 Source.plot        df2 aefficiency mefficiency eefficiency dforthog
##  Int              1 Blocks               1      0.6747      0.6747      0.6747        0
##  Days[Int]       14 Blocks               1      0.0166      0.0166      0.0166        0
##                     WRows[Blocks]       13      0.6754      0.7049      0.4519        0
##  Locn            27 Blocks               1      0.0201      0.0201      0.0201        0
##                     WRows[Blocks]       26      0.0435      0.1162      0.0092        0
##  Int#Locn        27 Blocks               1      0.0195      0.0195      0.0195        0
##                     WRows[Blocks]       26      0.0361      0.1075      0.0081        0
##  Days#Locn[Int] 378 Blocks               1      0.2691      0.2691      0.2691        0
##                     WRows[Blocks]       58      0.4676      0.7232      0.0955        0
##                     Cols                11      0.7837      0.7919      0.6586        0
##                     Blocks#Cols         11      0.7735      0.7825      0.6263        0
##                     WRows#Cols[Blocks] 297      0.4627      0.8988      0.0153      256
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source             df Alias                    In             aefficiency mefficiency eefficiency
##  Cols                1 Blocks                   plot                0.0136      0.0136      0.0136
##  Cols               11 WRows[Blocks]            plot                0.0569      0.0776      0.0257
##  Cols               11 ## Information remaining plot                0.9195      0.9213      0.8491
##  Blocks#Cols        22 WRows[Blocks]            plot                0.0424      0.0778      0.0076
##  Blocks#Cols        11 ## Information remaining plot                1.0000      1.0000      1.0000
##  WRows[Blocks]       1 Blocks                   Days[Int]           1.0000      1.0000      1.0000
##  WRows[Blocks]       1 Blocks                   Locn                1.0000      1.0000      1.0000
##  WRows[Blocks]       1 Blocks                   Int#Locn            1.0000      1.0000      1.0000
##  WRows[Blocks]       1 Blocks                   Days#Locn[Int]      0.2131      0.2131      0.2131
##  Cols                1 Blocks                   Days#Locn[Int]      0.0199      0.0199      0.0199
##  Cols               11 WRows[Blocks]            Days#Locn[Int]      0.0328      0.0478      0.0128
##  Blocks#Cols         1 Blocks                   Days#Locn[Int]      0.0135      0.0135      0.0135
##  Blocks#Cols        11 WRows[Blocks]            Days#Locn[Int]      0.0329      0.0521      0.0123
##  Blocks#Cols        11 Cols                     Days#Locn[Int]      0.0001      0.0064      0.0000
##  WRows#Cols[Blocks]  1 Blocks                   Days#Locn[Int]      0.5986      0.5986      0.5986
##  WRows#Cols[Blocks] 58 WRows[Blocks]            Days#Locn[Int]      0.0727      0.5100      0.0056
##  WRows#Cols[Blocks] 11 Cols                     Days#Locn[Int]      0.4438      0.5111      0.2140
##  WRows#Cols[Blocks] 11 Blocks#Cols              Days#Locn[Int]      0.4566      0.5128      0.2721
##  dforthog
##         0
##         0
##         0
##         0
##        11
##         1
##         1
##         1
##         0
##         0
##         0
##         0
##         0
##         0
##         0
```

```
##         5
##         0
##         0
##
## The design is not orthogonal
```

4.4.6 Substituting a linear Locations term for arbitrary Locations differences

```
#'## Substituting xLocn for Locations (and pooling Blocks and WRows to reduce the table)
ph2sys.odw.lin.canon <- designAnatomy(formulae   = list(lab = ~ IntDays + xLocn + IntDays:Locn,
                                                        plot = ~ (Rows*Cols)/Samp,
                                                        trt  = ~ Genotypes),
                                      keep.order = TRUE, data = layout)

## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Cols and Rows are partially
aliased in IntDays#Locn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Rows#Cols and Rows are
partially aliased in IntDays#Locn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Rows#Cols and Cols are
partially aliased in IntDays#Locn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Samp[Rows:Cols] and Rows
are partially aliased in IntDays#Locn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Samp[Rows:Cols] and Cols
are partially aliased in IntDays#Locn
## Warning in projs.2canon(CombinedSets$Q[[ntiers]], struct[[ktier]]$Q): Samp[Rows:Cols] and Rows#Cols
are partially aliased in IntDays#Locn

print(summary(ph2sys.odw.lin.canon, which.criteria = c("ae", "me", "ee", "dfor")))

##
##
## Summary table of the decomposition for lab, plot & trt (based on adjusted quantities)
##
##  Source.lab   df1 Source.plot     df2 Source.trt df3 aefficiency mefficiency eefficiency dforthog
##  IntDays       15 Rows             15 Genotypes   15      0.8069      0.8143      0.6355        0
##  xLocn          1 Rows              1 Genotypes    1      0.8206      0.8206      0.8206        0
##  IntDays#Locn 431 Rows             59 Genotypes   59      0.8217      0.8589      0.4394        6
##                   Cols             11 Genotypes   11      0.8976      0.9000      0.8127        0
##                   Rows#Cols       336 Genotypes  336      0.4091      0.8791      0.0145      283
##                   Samp[Rows:Cols]  25                      1.0000      1.0000      1.0000       25
##
## Table of information (partially) aliased with previous sources derived from the same formula
##
##  Source          df Alias                    In           aefficiency mefficiency eefficiency
##  Cols            11 Rows                     plot              0.0584      0.0787      0.0257
##  Cols            11 ## Information remaining plot              0.9195      0.9213      0.8491
##  Cols            11 Rows                     IntDays#Locn      0.0035      0.0235      0.0005
##  Rows#Cols       16 Rows                     IntDays#Locn      0.5040      0.5609      0.2508
##  Rows#Cols       11 Cols                     IntDays#Locn      0.0003      0.0050      0.0000
##  Samp[Rows:Cols] 16 Rows                     IntDays#Locn      0.1552      0.2581      0.0380
##  Samp[Rows:Cols] 11 Cols                     IntDays#Locn      0.0001      0.0014      0.0000
##  Samp[Rows:Cols] 16 Rows#Cols                IntDays#Locn      0.0088      0.0419      0.0022
##  dforthog
```

```
##         0
##         0
##         0
##         0
##         0
##         0
##         0
##         0
##
## The design is not orthogonal
```

4.4.7 Questions

1. What permutations are performed in randomizing the systematic p/q-rep design¿

   Because of the setting of the nested.recipients argument in designRandomize, the Intervals are permuted, the Days within Intervals are permuted and the Locations within each Intervals-Days combinations are permuted.

2. Where is most of the information about Rows confounded in the two-phase design?

   From the anatomy for the combined-units design, the largest amount of information (64.7%) about Blocks is confounded with Intervals and a further 28.5% is confounded with Days[Intervals]. Also, a large amount (74.5%) of the information about WRows[Blocks] is confounded Days[Intervals]. There is not much information about Blocks and WRows[Blocks] confounded with other milling phase sources.

3. What are the effects on the analysis of being able to describe the Locations differences in terms of a linear trend term instead of arbitrary differences between Locations?

   *From the anatomy in which xLocn is substituted for Locations, the df for Genotypes estimable from Rows#Cols has increased substantially from 297 to 335. Also the mefficiency for Genotypes estimable from Rows#Cols has increased from 0.8192 to 0.8825. That is the amount of information about all Genotypes information confounded with Rows#Cols has increased from 0.6594 (= 0.8192*297/369) to 0.8012 (= 0.8825*335/369). Also, there is now available 25 df for Samples[Rows´Cols] when combined with Intervals#Days#Locations, i.e. 25 Error df.*

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*(3), 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: https://cran.at.r-project.org/package=asremlPlus/, (R package version 4.3.45, accessed January 20, 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 3.2.14, accessed January 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 C. G. B. Demétrio (1998) Using the randomisation in specifying the ANOVA model and table for properly and improperly replicated grazing trials. *Australian Journal of Experimental Agriculture*, *38*, 325–334.

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. *Biometrical Journal*, *??*, ???–???

Butler, D. G. (2022) odw: Generate optimal experimental designs. R package version 2.1.4.

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.

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.

Kaps, M. and W. Lamberson (2004) *Biostatistics for animal science*. Wallingford, UK: CABI Publishing.

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

Mead, R. (1990) *The Design of Experiments*. Cambridge: Cambridge University Press.

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>.

Roberts, E. A. (1975) Accounting for components of error. In V. Bofinger and J. Wheeler (Eds.), *Developments in Field Experiment Design and Analysis*. Commonwealth Bureau of Pastures and Field Crops, Chapter 5, pp. 59–71. Slough: Commonwealth Agricultural Bureaux.

Sermarini, R. A., C. Brien, C. G. B. Demétrio, and A. dos Santos (2020) Impact on genetic gain from using misspecified statistical models in generating p-rep designs for early generation plant-breeding experiments. *Crop Science*, *60*(6), 3083–3095.

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*.