Obtaining, randomizing, exposing the confounding & formulating mixed models for designs for comparative experiments using **R**

C. J. Brien, R. A Sermarini and C. G. B. Demétrio

May 18, 2024

Short course, RBras 68, Piracicaba, Brasil, 28th May 2024

This document describes how to use functions from the R (R Core Team, 2024) packages `dae` (Brien, 2024b) 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 (4.2.x or later preferable)
- RStudio
- Packages (you can check the version using the `packageVersion` function.)
  - dae (Version 3.2.24 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

08:30–09:30: Topic I. Concepts in experimental design & analysis: 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.

09:30–10:30: Practical I. Orthogonal experimental design & analysis in R : using dae to generate orthogonal designs for experiments.

10:30–11:00: Refreshment break

11:00–12:00: Topic II. Nonorthogonal experimental design & analysis: 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.

12:00–13:00: Practical II. Nonorthogonal experimental design & analysis in R : using dae and odw to produce a range of nonorthogonal designs for experiments.

13:00–14:30: Lunch

14:00–15:30: Topic III. Miscellaneous topics in experimental design & analysis: systematic allocation, pseudoreplication, block-treatment interactions, experiments involving time, and nested factorials.

15:30–16:00: Refreshment break

16:00–17:00: Practical III. Miscellaneous topics in experimental design & analysis in R : further use of dae and odw related to the miscellaneous topics.

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, 2024b). 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 has 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 random effects for each observation, sometimes referred to as random errors. 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 that specify a different effect for each observation are called identity terms and are underlined.

Topic I. Orthogonal experimental design & analysis 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 units and treatments factors, respectively.

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

```
library(dae)
packageVersion("dae")
```

I.1 Two potential designs for a 5×5 grid of plots (Brien et al., 2023, Section 2)

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.

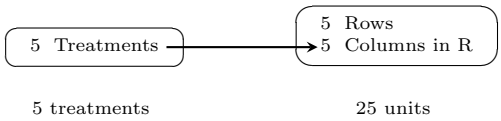

Figure 1: Factor-allocation diagram for an RCBD: treatments are allocated to units; the arrow indicates that the factor Treatments is randomized to Columns; Columns in R indicates that the Columns are considered to be nested within Rows for this randomization; R = Rows.

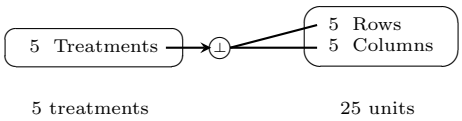

Figure 2: Factor-allocation diagram for an LSqD: treatments are allocated to units; the arrow indicates that the allocation is randomized; the '⊥' at the end of the arrow indicates that an orthogonal design is used; the two lines from '⊥' indicates that the Treatments are allocated to the combinations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

I.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 several 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 `factor`s, 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 `factor`s in standard order and often will also be used to generate the values of the allocated `factor`s.

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 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
#'## 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.

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

I.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(units = ~ Rows/Columns,
                                             trts  = ~ Treatments),
                            grandMean = TRUE, data = RCBD.lay)
summary(RCBD.canon)
#'## Anatomy for the LSqD
LSqD.canon <- designAnatomy(formulae  = list(units = ~ Rows*Columns,
                                             trts  = ~ Treatments),
                            grandMean = TRUE, data = LSqD.lay)
summary(LSqD.canon)
```

I.1.4 Questions

1. What is the advantage of specifying a `seed` in `designRandomize`?
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?
3. What would determine which of these two designs is used for a particular experiment?

I.2 Split-unit design for a pasture experiment 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.

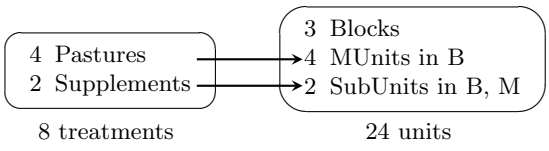

Figure 3: Factor-allocation diagram for a split-unit design: treatments are allocated to units; the arrows indicates that the factors Pastures and Supplements are randomized to MUnits and SubUnits, respectively; MUnits in B indicates that the MUnits are considered to be nested within Blocks for this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

I.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:

```
#'## 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 with blue lines showing MainUnits, first combining Pastures and
#'     Supplements so that they are plotted 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(units = ~ Blocks/MUnits/SubUnits,
                                             trts  = ~ Pastures*Supplements),
                            grandMean = TRUE, data = Milk.lay)
summary(Milk.canon, which.criteria = c("aeff", "order"))
```

I.2.2 Questions

1. In what sense does this design involve a single randomization?
2. What is the initial allocated mixed model for this design? Is it equivalent to a randomization model?
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?

I.3 Split-unit design for an experiment in which time is randomized (Brien et al., 2023, Section 4.1)

A design is required for a conventional greenhouse experiment to investigate Zinc effects on plants of a medic species. A response over five weeks is to be measured, but the measurement of the response requires destructive harvesting of the plants. It is to involve four levels of Zinc and there are to be eight replicates of the Zinc-Weeks combinations. The Weeks are assigned to the main units, formed from 4 pots, using a randomized complete-block design with 8 blocks and the Zinc levels are randomly assigned to the pots in each main unit. The factor-allocation diagram for the experiment is in Figure 4.

I.3.1 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 R code below. Note that this experiment has the additional requirement that the design be located in the greenhouse.

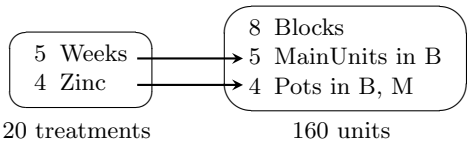

Figure 4: Factor-allocation diagram for a split-unit design: treatments are allocated to units; the arrows indicates that the factors Weeks and Zinc are randomized to MainUnits and Pots, respectively; MainUnits in B indicates that the MainUnits are considered to be nested within Blocks for this randomization; Pots in B, M indicates that the Pots are considered to be nested within Blocks and MainUnits for this randomization; B = Blocks, M = MainUnits

```
#'## Set up the systematic design
SUD.sys <- cbind(fac.gen(list(Blocks = 8, MainUnits = 5, Pots = 4)),
                 fac.gen(list(Weeks = LETTERS[1:5], Zinc = 4), times = 8))

#'## Obtain the randomized layout
SUD.lay <- designRandomize(allocated         = SUD.sys[c("Zinc", "Weeks")],
                           recipient         = SUD.sys[c("Blocks", "MainUnits", "Pots")],
                           nested.recipients = list(MainUnits = "Blocks",
                                                    Pots = c("MainUnits", "Blocks")),
                           seed = 3116)
#'## Locate the design in the glasshouse and plot
SUD.lay <- cbind(SUD.lay,
                 with(SUD.lay, fac.divide(Pots, list(PLane = 2, PPosn = 2))),
                 with(SUD.lay, fac.divide(Blocks, list(BLane = 2, BPosn = 4))))
SUD.lay <- within(SUD.lay,
                  {
                    Lanes <- fac.combine(list(BLane, PLane))
                    Positions <- fac.combine(list(BPosn, MainUnits, PPosn))
                    Treatments <- fac.combine(list(Weeks,Zinc), combine.levels = TRUE)
                  })
SUD.lay <- SUD.lay[c("Lanes", "Positions", "Blocks", "MainUnits", "Pots",
                     "Zinc", "Weeks", "Treatments")]
designGGPlot(SUD.lay, labels = "Treatments", label.size = 5,
             row.factors = "Positions", column.factors = "Lanes",
             cellfillcolour.column = "Weeks",
             title = NULL, title.size = 18, axis.text.size = 15,
             blockdefinition = cbind(10,2))

#'## Check its properties
SUD.canon <- designAnatomy(formulae = list(units = ~ Blocks/MainUnits/Pots,
                                           trts  = ~ Zinc*Weeks),
                           grandMean = TRUE, data = SUD.lay)
summary(SUD.canon)
```

I.3.2 Questions

1. What is the initial allocated mixed model for this design? Is it equivalent to a randomization model?
2. In general terms, how are the two Residual mean squares expected to compare in magnitude?

I.4 Split-unit design with criss-cross design of the sub-units 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 × 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`.

I.4.1 Questions

1. Describe the confounding that is inherent in this design.
2. Draw a factor-allocation diagram for this experiment.
3. What is the initial allocated mixed model for this design?

I.5 A design for the petrol additives experiment

Box et al. (2005, Section 4.4) describe 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 5.

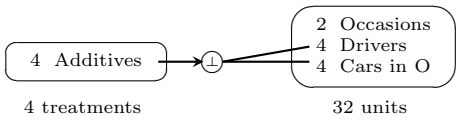

Figure 5: Factor-allocation diagram for repeated LSqDs: treatments are allocated to units; the arrow indicates that the allocation is randomized; the '①' at the end of the arrow indicates that an orthogonal design is used; the two lines from '①' indicates that the Additives are allocated to the combinations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

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

#'## 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(units = ~ (Occasions/Cars)*Drivers,
                                                   trts  = ~ Additives),
```

```
                                    grandMean = TRUE, data = LSRepeat2b.lay)
summary(LSRepeat2b.canon)
```

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 6.

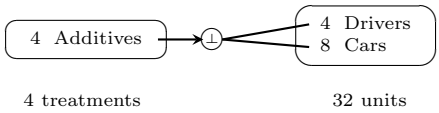

Figure 6: Factor-allocation diagram for repeated LSqDs: treatments are allocated to units; the arrow indicates that the allocation is randomized; the '⊥' at the end of the arrow indicates that an orthogonal design is used; the two lines from '⊥' indicates that the Additives are allocated to the combinations of Drivers and Cars using the design.

```
#'## 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(units = ~ Drivers*Cars,
                                                     trts  = ~ Additives),
                                    grandMean = TRUE, data = LSRepeat2b.D8.lay)
summary(LSRepeat2.D8.canon)

#'## Add Occasions to the analysis
LSRepeat2b.D8.lay$Occasions <- fac.recast(LSRepeat2b.D8.lay$Cars,
                                          newlevels = rep(1:2, each=4))
LSRepeat2b.D8.lay
LSRepeat2b.D8.canon <- designAnatomy(formulae  = list(units = ~ (Occasions + Cars)*Drivers,
                                                      trts  = ~ Additives),
                                     grandMean = TRUE, data = LSRepeat2b.D8.lay)
summary(LSRepeat2b.D8.canon)
```

I.5.1 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?
2. What is the difference between the two randomizations?
3. How do the two anatomies that include Occasions differ?
4. What effect does including Occasions#Drivers have on the anatomy?

Topic II. Nonorthogonal experimental design & analysis 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. **?**, Section 5 discuss the anatomy and its interpretation for a nonorthogonal, balanced experiment.

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

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

II.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	2	3	4	5	1	2	3	4	5	1	2	3	4	5
	6	7	8	9	10	7	8	9	10	6	8	9	10	6	7
	11	12	13	14	15	13	14	15	11	12	15	11	12	13	14
	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 7.

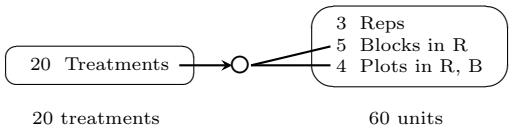

Figure 7: Factor-allocation diagram for the alpha design: treatments are allocated to units; the arrow indicates that the allocation is randomized; the 'O' at the end of the arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicate that the Treatments are allocated to the combinations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

II.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 randomized 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",
                             grandMean      = TRUE, data = alpha.lay)
summary(alpha.canon, which.criteria = "all")
```

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[4]
(measures <- c(AVPD, Aeff, 2/(3*Aeff)))
```

II.1.2 Questions

1. What is the randomization-based mixed model for this experiment?
2. In a mixed-model analysis, which unit terms might you fit as fixed terms? Why?

II.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 8.

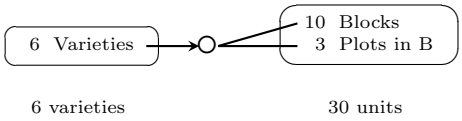

Figure 8: Factor-allocation diagram for the balanced incomplete-block design: treatments are allocated to units; the arrow indicates that the allocation is randomized; the 'O' at the end of the arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicates that the Varieties are 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.

II.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),
                                 grandMean = TRUE, data = BIBDWheat.dat)
summary(bibdwheat.canon)
```

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[4]
(measures <- c(AVPD, Aeff, 2/(5*Aeff)))
```

II.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, label.size = 6, blockdefinition = cbind(1,3))
bibdwheat.Miss.canon <- designAnatomy(formulae  = list(units = ~ Blocks/Plots,
                                                       trts  = ~ Varieties),
                                      grandMean = TRUE, data = na.omit(bibdwheat.Miss.dat))
summary(bibdwheat.Miss.canon, which.criteria = c('aeff', "xeff", "eeff", 'order'))
```

II.2.3 Questions

1. What is the value of xefficiency for Varieties when confounded with Plots[Blocks] for the original design? Why?

2. How many nonzero eigenvalues does $\mathbf{Q}_{\mathrm{V}}\mathbf{Q}_{\mathrm{BP}}\mathbf{Q}_{\mathrm{V}}$ have?

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

II.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 9.

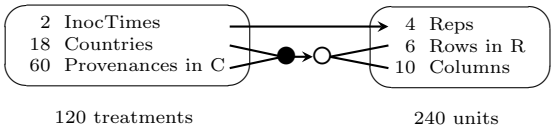

Figure 9: Factor-allocation diagram for the balanced lattice design: treatments are allocated to units; the arrows indicates that the allocations are randomized; the two lines leading to the '●' indicate that it is the combinations of Countries and Provenances that is allocated; the '○' at the end of the arrow from the '●' 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.

II.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)),
                                 grandMean = TRUE, data = Casuarina.dat)
summary(Casuarina.canon, which = c("aeff", "eeff", "order", "dforth"))
```

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.

II.3.2 Questions

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

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

II.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 10.

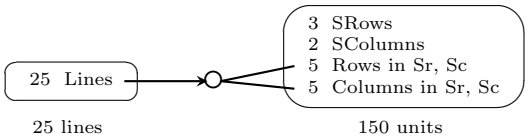

Figure 10: Factor-allocation diagram for the balanced lattice square design: treatments are allocated to units; the arrows indicates that the allocations are randomized; the 'O' at the end of the lower arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicates that the Lines 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.

II.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 (2024a) 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)
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, label.size = 6,
             blockdefinition = cbind(5,5))
#'## Check the properties of the design
latt.canon <- designAnatomy(formulae  = list(units = ~ (SRows:SColumns)/(Rows*Columns),
```

```
                                              trts  = ~ Variety),
                            grandMean = TRUE, data = latt.lay)
summary(latt.canon, which.criteria = c("aeff", "order"))
```

II.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)
#'### 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)
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.

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

II.4.4 Questions

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

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

Topic III. Miscellaneous topics in experimental design & analysis 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")
options(width=100)
```

III.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 11.

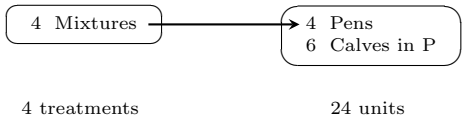

Figure 11: Factor-allocation diagram for the animal feeding experiment: treatments are allocated to units; the arrow indicates that the factor Mixtures is randomized to Pens; Calves in P indicates that the Calves are nested within Pens; P = Pens.

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

```
#'## Set up the systematic design and obtain the randomized 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, label.size = 6, blockdefinition = cbind(1,6))

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

III.1.1 Questions

1. How is the pseudoreplication involved in this experiment manifested in the anatomy? (Brien et al. (2023, Section 4.3) discuss the issues associated with pseudoreplication.)

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?

III.2 Block-treatment interactions for an experiment in which time is randomized (Brien et al., 2023, Section 4.1)

The properties of a split-unit design have been examined in Section I.3 for an experiment in which the effects on Zinc over five weeks were investigated. In that investigation, the terms in the initial allocation model were considered. Here the properties of a homogeneous model with block-treatment interactions are checked. If you have not saved the design, reconstruct it as shown below; otherwise, use the saved design. Then, obtain the anatomy to establish its properties.

```
#'## Set up the systematic design
SUD.sys <- cbind(fac.gen(list(Blocks = 8, MainUnits = 5, Pots = 4)),
                 fac.gen(list(Weeks = LETTERS[1:5], Zinc = 4), times = 8))

#'## Obtain the randomized layout
SUD.lay <- designRandomize(allocated         = SUD.sys[c("Zinc", "Weeks")],
                           recipient         = SUD.sys[c("Blocks", "MainUnits", "Pots")],
                           nested.recipients = list(MainUnits = "Blocks",
                                                    Pots = c("MainUnits", "Blocks")),
                           seed = 3116)

#'## Check its properties
SUD.BT.canon <- designAnatomy(formulae = list(units = ~ Blocks/MainUnits/Pots,
                                              trts  = ~ Blocks*Zinc*Weeks),
                              grandMean = TRUE, data = SUD.lay)
summary(SUD.BT.canon)
```

III.2.1 Questions

1. What do you conclude from the anatomy about the estimability of terms?

2. How might you change the design so that the block-treatment interactions are separately estimable?

III.3 A longitudinal greenhouse experiment that uses a generalized randomized block design (GRBD)(Brien et al., 2023, Section 4.2)

Consider an experiment in a glasshouse that has equipment to automatically image plants daily, the images being processed to produce a measure related to plant biomass. Suppose that the experiment to investigate the effects of four levels of Zinc on medic plants is to be run in this glasshouse and that the plants are to be imaged over 14 Days. It is to involve 12 replicates and the experimental area can accommodate 48 pots in grid of four lanes by 12 positions, each pot having a single plant. Previous experience is that the differences between pots in the same lane separated by more than two pots are likely to be larger than between those separated by no more than two pots. Also, pots in the front pair of lanes are likely to differ from pots in the back pair of lanes. That is, Blocks consisting of eight pots arranged in two lanes by four positions are likely to be relatively homogeneous. Further, suppose that it is thought that the response to Zinc may differ between the Blocks. As Brien et al. (2023, Section 3.2) conclude, a GRBD is a suitable design for this experiment.

The factor allocation diagram for this experiment is given in Figure 12

Obtain the randomized layout for this longitudinal experiment.

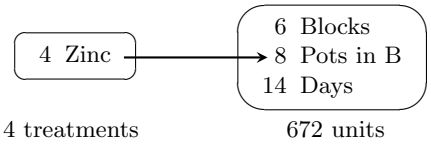

Figure 12: Factor-allocation diagram showing the treatments allocation to units for the longitudinal experiment that uses a generalized randomized block design: the arrow indicates that Zinc is allocated to Pots within B using randomization; B = Blocks.

```
#'## Construct a systematic design
longi.sys <- cbind(fac.gen(list(Blocks = 6, Pots = 8, Days = 14)),
                   fac.gen(list(Zinc = LETTERS[1:4]), times = 12, each = 14))

#'## Obtain the randomized layout
longi.lay <- designRandomize(allocated         = longi.sys["Zinc"],
                             recipient         = longi.sys[c("Blocks", "Pots", "Days")],
                             nested.recipients = list(Pots = "Blocks"),
                             seed = 5733)

#'## Add factors for Lane and Position
longi.lay <- cbind(with(longi.lay, fac.divide(Blocks,
                                              factor.names = list(PLanes = 2,
                                                                  QPositions = 3))),
                   with(longi.lay, fac.divide(Pots,
                                              factor.names = list(Lanes = 2,
                                                                  Positions = 4))),
                   longi.lay)
longi.lay <- within(longi.lay,
                    {
                      Lanes <- fac.combine(list(PLanes, Lanes))
                      Positions <- fac.combine(list(QPositions, Positions))
                    })
longi.lay <- longi.lay[, -match(c("PLanes", "QPositions"), names(longi.lay))]

#'## Plot the layout
designGGPlot(subset(longi.lay, Days == "1"),
             row.factors = "Lanes", column.factors = "Positions",
             labels = "Zinc", label.size = 8,
             title = NULL, title.size = 25, axis.text.size = 20,
             blockdefinition = cbind(2,4))
```

The homogeneous allocation model given by Brien et al. (2023, Section 4.2) is:

$$\text{Mean} + \text{Z} + \text{D} + \text{Z}\wedge\text{D} \quad | \quad \text{Mean} + \text{B} + \text{B}\wedge\text{Z} + \text{B}\wedge\text{P} + \text{B}\wedge\text{D} + \text{B}\wedge\text{Z}\wedge\text{D} + \underline{\text{B}\wedge\text{P}\wedge\text{D}}.$$

Check the properties of the layout corresponding to the homogeneous allocation model using an anatomy.

```
longi.BZD.canon <- designAnatomy(formulae = list(units   = ~ (Blocks/Pots)*Days,
                                                 trtblks = ~ Blocks*Zinc*Days),
                                 grandMean = TRUE, data = longi.lay)
summary(longi.BZD.canon)
```

III.3.1 Questions

1. What are the block-treatments interactions in this experiment? Are they all estimable?

2. How do the properties of the longitudinal experiment differ from those of the experiment in which Weeks are randomized (Sections I.3 and III.2)?

III.4 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 13.

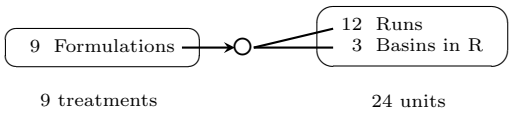

Figure 13: Factor-allocation diagram for the detergent experiment: treatments are allocated to units; the arrow indicates that the allocation is randomized; the 'O' at the end of the arrow indicates that a nonorthogonal design is used; the two lines from 'O' indicate that the Treatments are allocated to the combinations of 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.

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

III.4.1 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,
                                          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(units = ~ Runs/Basins,
                                             trts  = ~ Formulations),
                            grandMean = TRUE, data = BIBD.lay)
summary(BIBD.canon, which.criteria = c('aeff', 'order'))
```

III.4.2 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(units = ~ Runs/Basins,
                                                  trts  = ~ Types/(Bases*Additives)),
                                 grandMean = TRUE, data = BIBD.lay)
summary(BIBD.nest.canon, which.criteria = c('aeff', 'order'))
```

III.4.3 Leave out Types and try decomposition with Bases and Additives in both orders

```
BIBD.nest2.canon <- designAnatomy(formulae  = list(units = ~ Runs/Basins,
                                                   trts  = ~ Bases*Additives),
                                  grandMean = TRUE, data = BIBD.lay)
```

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

III.4.4 Questions

1. What do you conclude about the properties of the design both without and with the nested factors?
2. What is the effect of removing the Types factor?
3. What is the advantage of using nested factors for this experiment?
4. Is there any reason to think that a row-column design might be better than a block design for this experiment?

III.5 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

#'## Plot the layout
#+ "RCBDSpray_v1"
designGGPlot(RCBD.lay, labels = "Treatments",
             cellfillcolour.column = "Rates",
             row.factors = "Blocks", column.factors = "Plots",
             axis.text.size = 20, label.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(units = ~ Blocks/Plots,
                                             trts  = ~ Rates/(Rate2090 + Rate2930 + Rate4120 +
                                                               Rate5770)),
                            grandMean = TRUE, data = RCBD.lay)
summary(RCBD.canon, which.criteria = "aeff")
```

III.5.1 Questions

1. What is the prior allocation model for this design?
2. How does the prior allocation model differ from the randomization model for this design?
3. Why are terms involving Rate2090 and Rate5770 not included in the prior allocation model?

III.6 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 units structure
PBIBD.lay <- designRandomize(allocated         = PBIBD.sys["Treatments"],
                             recipient         = PBIBD.sys[c("Blocks", "Units")],
                             nested.recipients = list(Units = "Blocks"),
                             seed              = 65460)
PBIBD.lay

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

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$Control <- with(PBIBD.lay, fac.uselogical(Treatments == "A",
                                                    labels = c("Control", "rest")))
PBIBD.canon <- designAnatomy(formulae  = list(units = ~ Blocks/Units,
                                              trts  =  ~ Control + Treatments),
                             grandMean = TRUE, data = PBIBD.lay)
summary(PBIBD.canon, which.criteria = c('aeff', 'xeff', 'eeff','order', 'dforth'))
#'### Try other treatments
PBIBD.lay$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)
PBIBD.odw.lay <- PBIBD.odw$design
```

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

```
#'### Randomize design according to the units 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

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

1. Why must the Control source be balanced?
2. How do the Cochran and Cox design and the design obtained with odw compare?

III.7 The Casuarina experiment (continued)

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

```
#'## 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,
                                 grandMean  = TRUE,
                                 data       = Casuarina.dat)
summary(Casuarina.canon, which = c("aeff", "eeff", "order", "dforth"))
```

III.7.1 Questions

1. How does this analysis compare with that conducted in Section II.3?

References

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

Brien, C. J. (2024a) *asremlPlus: Augments 'ASReml-R' in Fitting Mixed Models and Packages Generally in Exploring Prediction Differences*. URL http://CRAN.R-project.org/package=asremlPlus/, (R package version 4.4.32, accessed May 17, 2024).

Brien, C. J. (2024b) *dae: functions useful in the design and ANOVA of experiments*. URL http://CRAN.R-project.org/package=dae/, (R package version 3.2.25, accessed May 17, 2024).

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. 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*, *65*(7), 2200284.

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.

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 (2024) *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. URL https://www.r-project.org/.

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

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