THE THEORY OF THE DESIGN OF EXPERIMENTS

D.R.Cox and N. Reid

Contents

Preface **7**

1 Some general concepts **1**
- 1.1 Types of investigation 1
- 1.2 Observational studies 3
- 1.3 Some key terms 4
- 1.4 Requirements in design 5
- 1.5 Interplay between design and analysis 6
- 1.6 Key steps in design 7
- 1.7 A simplified model 11
- 1.8 A broader view 11
- 1.9 Bibliographic notes 14
- 1.10 Further results and exercises 15

2 Avoidance of bias **19**
- 2.1 General remarks 19
- 2.2 Randomization 19
- 2.3 Retrospective adjustment for bias 29
- 2.4 Some more on randomization 32
- 2.5 More on causality 34
- 2.6 Bibliographic notes 36
- 2.7 Further results and exercises 37

3 Control of haphazard variation **41**
- 3.1 General remarks 41
- 3.2 Precision improvement by blocking 42
- 3.3 Matched pairs 43
- 3.4 Randomized block design 48
- 3.5 Partitioning sums of squares 53
- 3.6 Retrospective adjustment for improving precision 57

3.7 Special models of error variation 61
3.8 Bibliographic notes 62
3.9 Further results and exercises 62

4 Specialized blocking techniques 65
4.1 Latin squares 65
4.2 Incomplete block designs 70
4.3 Cross-over designs 85
4.4 Bibliographic notes 95
4.5 Further results and exercises 95

5 Factorial designs: basic ideas 99
5.1 General remarks 99
5.2 Example 101
5.3 Main effects and interactions 102
5.4 Example: continued 109
5.5 Two level factorial systems 110
5.6 Fractional factorials 116
5.7 Example 120
5.8 Bibliographic notes 122
5.9 Further results and exercises 123

6 Factorial designs: further topics 127
6.1 General remarks 127
6.2 Confounding in 2^k designs 127
6.3 Other factorial systems 131
6.4 Split plot designs 140
6.5 Nonspecific factors 144
6.6 Designs for quantitative factors 149
6.7 Taguchi methods 157
6.8 Conclusion 160
6.9 Bibliographic notes 162
6.10 Further results and exercises 163

7 Optimal design 169
7.1 General remarks 169
7.2 Some simple examples 169
7.3 Some general theory 173
7.4 Other optimality criteria 176
7.5 Algorithms for design construction 177

7.6 Nonlinear design 178
7.7 Space-filling designs 181
7.8 Bayesian design 182
7.9 Optimality of traditional designs 186
7.10 Bibliographic notes 186
7.11 Further results and exercises 188

8 Some additional topics 193
8.1 Scale of effort 193
8.2 Adaptive designs 201
8.3 Sequential regression design 208
8.4 Designs for one-dimensional error structure 209
8.5 Spatial designs 215
8.6 Bibliographic notes 219
8.7 Further results and exercises 221

A Statistical analysis 225
A.1 Introduction 225
A.2 Linear model 226
A.3 Analysis of variance 238
A.4 More general models; maximum likelihood 244
A.5 Bibliographic notes 244
A.6 Further results and exercises 245

B Some algebra 249
B.1 Introduction 249
B.2 Group theory 249
B.3 Galois fields 254
B.4 Finite geometries 258
B.5 Difference sets 260
B.6 Hadamard matrices 261
B.7 Orthogonal arrays 262
B.8 Coding theory 263
B.9 Bibliographic notes 264
B.10 Further results and exercises 265

C Computational Issues 267
C.1 Introduction 267
C.2 Overview 268
C.3 Randomized block experiment from Chapter 3 274

C.4 Analysis of block designs in Chapter 4 281
C.5 Examples from Chapter 5 287
C.6 Examples from Chapter 6 297
C.7 Bibliographic notes 303

References 305

List of tables 317

Author index 319

Index 323

APPENDIX C

Computational Issues

Revised and converted to R by Wei Lin and Nancy Reid, July 2010.

C.1 Introduction

In the published version of the book (Chapman & Hall, 2000), Appendix C included code in S-PLUS for the examples discussed in the text. In this addendum we provide an updated and corrected version of this Appendix, with all the code converted to R. The examples in this supplement were run under R version 2.11.1. For ease of comparison with the original version we have kept the text largely the same, except in this Introduction, or where R-specific functions are introduced.

There is a wide selection of statistical computing packages, and most of these provide the facility for analysis of variance and estimation of treatment contrasts in one form or another. With small data sets it is often straightforward, and very informative, to compute the contrasts of interest by hand. In 2^k factorial designs this is easily done using Yates's algorithm (Exercise 5.1).

R is an open-source statistical language and environment modeled after S and its commercial implementation, S-PLUS. It is freely available under a GNU General Public License. Originally created by Robert Gentleman and Ross Ihaka at the University of Auckland in 1995, it is now maintained by the R Development Core Team[*] through the R Foundation, and is very widely used in the statistics community. A great strength of R is the large number of packages that can be installed as add-ons to the basic distributions. Software and packages can be downloaded from the R project website `http://www.r-project.org/`.

We give here a very brief overview of the analysis of the more

[*] R Development Core Team (2007). *A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing; Vienna, Austria.

standard designs using R, by providing code sufficient for the analysis of the main examples in the text. The reader needing an introduction to R or wishing to exploit its full capabilities will need to consult one of the several books on the topic. We have found Faraway,[†] Maindonald & Braun[‡] and Venables & Ripley[§] to be good general references; see the Bibliographic Notes for references explicitly for design of experiments. As with many statistical packages, the output from R is typically not in a form suitable for the presentation of conclusions, an important aspect of analysis that we do not discuss.

We assume the reader is familiar with running R on the system being used and with the basic structure of R, including data manipulation and the use of functions, as well as the use of *objects* and *methods* for objects. A dataset, a fitted regression model, and a residual plot are all examples of objects. Examples of methods for these objects are `summary`, `plot` and `residuals`. Many objects have several specific methods for them as well; for example `lm.influence` computes diagnostics for a fitted linear model object. The illustrations below use a command line version of R; a menu-driven version, R-commander, is also available.[¶]

C.2 Overview

C.2.1 Data entry

The typical data from the types of experiments we describe in this book takes a single response or dependent variable at a time, several classification variables such as blocks, treatments, factors and so on, and possibly one or more continuous explanatory variables, such as baseline measurements. The dependent and explanatory variables will typically be entered from a terminal or file, using a version of the `scan` or `read.table` function. It will rarely be the case that the data set will contain fields corresponding to the various classification factors. These can usually be constructed us-

[†] Faraway, J.J. (2004). *Linear Models with R*, CRC Press, Boca Raton.

[‡] Maindonald, J. and Braun, W.J. (2003). *Data Analysis and Graphics Using R: An Example-based Approach*, Cambridge University Press, Cambridge

[§] Venables, W.N. and Ripley, B.D. (2002). *Modern Applied Statistics with S*, Springer-Verlag, New York.

[¶] `http://socserv.socsci.mcmaster.ca/jfox/Misc/Rcmdr/`

ing the `rep` function. All classification or factor variables must be explicitly declared to be so using the `factor` function.

Classification variables for full factorial designs can be created using `fac.design` in the package `DoE.base`, or for 2-level designs using `ffDesMatrix` in the package `BHH2`.

The collection of explanatory, baseline, and classification variables can be referred to in a variety of ways. The simplest, though in the long run most cumbersome, is to note that variables are automatically saved in the current working directory by the names they are assigned as they are read or created. In this case the data variables relevant to a particular analysis will nearly always be vectors with length equal to the number of responses. Alternatively, when the data file has a spreadsheet format with one row per case and one column per variable, it is often easy to store the dependent and explanatory variables as a matrix. The most flexible and ultimately powerful way to store the data is as a `data.frame`, which is essentially a matrix with rows corresponding to observations and columns corresponding to variables, and a provision for assigning names to the individual columns and rows.

In the first example below we illustrate these three methods of defining and referring to variables: as vectors, as a matrix, and as a data frame. In subsequent examples we always combine the variables in a data frame, using a design object for the explanatory variables if available.

As will be clear from the first example, one disadvantage of a data frame is that individual column must be accessed by the slightly cumbersome form `data.frame.name$variable.name`. One can refer to the variables in the data frame by their names alone by using the function `attach(data.frame)`.

C.2.2 Treatment means

The first step in an analysis is usually the construction of a table of treatment means. These can be obtained using the `tapply` function, illustrated in Section C.3 below. To obtain the mean response of y at each of several levels of x use `tapply(y, x, mean)`. In most of our applications x will be a factor variable, but in any case the elements of x are used to define categories for the calculation of the mean. If x is a list then cross-classified means are computed; we use this in Section C.5. In Section C.3 we illustrate the use of `tapply` on a variable, on a matrix, and on a data frame.

A data frame that contains a design object or a number of factor variables has several specialized plotting methods, the most useful of which is `interaction.plot`. Curiously, a summary of means of a design object does not seem to be available, although these means are used by the plotting methods for design objects.

An analysis of variance will normally be used to provide estimated standard errors for the treatment means, using the `aov` function described in the next subsection. If the design is completely balanced, the `model.tables` function can be used on the result of an `aov` function to construct a table of means after an analysis of variance, and this, while in principle not a good idea, will sometimes be more convenient than constructing the table of means before fitting the analysis of variance. For unbalanced or incomplete designs, `model.tables` will give estimated effects, but they are not always properly adjusted for lack of orthogonality.

C.2.3 Analysis of variance

Analysis of variance is carried out using the `aov` function, which is a specialization of the `lm` function used to fit a linear model. The summary and plot methods for `aov` are designed to provide the information most often needed when analysing these kinds of data.

The input to the `aov` function is a response variable and a model formula. R has a powerful and flexible modelling language which we will not discuss in any detail. The model formulae for most analyses of variance for balanced designs are relatively straightforward. The model formula takes the form `y ~ model`, where y is the response or dependent variable. Covariates enter `model` by their names only and an overall mean term (denoted `1`) is always assumed to be present unless explicitly deleted from the model formula. If A and B are factors `A + B` represents an additive model with the main effects of A and B, `A:B` represents their interaction, and `A*B` is shorthand for `A + B + A:B`. Thus the linear model

$$E(Y_{js}) = \mu + \beta x_{js} + \tau_j^A + \tau_s^B + \tau_{js}^{AB}$$

can be written

```
y~x+A*B
```

while

$$E(Y_{js}) = \mu + \beta_j x_{js} + \tau_j^A + \tau_s^B + \tau_{js}^{AB}$$

can be written

```
y~x+x:A+A*B.
```

There is also a facility for specifying nested effects; for example the model $E(Y_{a;j}) = \mu + \tau_a + \eta_{aj}$ is specified as `y ~ A+B/A`.

Model formulae are discussed in detail by Chambers and Hastie (1992, Chapter 2).

The analysis of variance table is printed by the `summary` function, which takes as its argument the name of the `aov` object. This will show sums of squares corresponding to individual terms in the model. The `summary` function does not show whether or not the sums of squares are adjusted for other terms in the model. In balanced cases the sums of squares are not affected by other terms in the model but in unbalanced cases or in more general models where the effects are not orthogonal, the interpretation of individual sums of squares depends crucially on the other terms in the model.

R computes the sums of squares much in the manner of stagewise fitting described in Appendix A, and it is also possible to update a fitted model using special notation described in Chambers and Hastie (1992, Chapter 2). The convention is that terms are entered into the model in the order in which they appear on the right hand side of the model statement, so that terms are adjusted for those appearing above it in the `summary` of the `aov` object. For example,

```
unbalanced.aov  <-  aov(y ~ x1 + x2 + x3); summary(unbalanced.aov)
```

will fit the models

$$\begin{aligned} y &= \mu + \beta_1 x_1 \\ y &= \mu + \beta_1 x_1 + \beta_2 x_2 \\ y &= \mu + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \end{aligned}$$

and in the partitioning of the regression sum of squares the sum of squares attributed to x_1 will be unadjusted, that for x_2 will be adjusted for x_1, and that for x_3 adjusted for x_1 and x_2. Be warned that this is not flagged in the output except by the order of the terms:

```
> summary(unbalanced.aov)
           Df  Sum of Sq  Mean Sq   F Value  Pr(F)
        x1     (unadj.)
        x2     (adj. for x1)
        x3     (adj. for x1,  x2)
 residuals
```

C.2.4 Contrasts and partitioning sums of squares

As outlined in Section 3.5, it is often of interest to partition the sums of squares due to treatments using linear contrasts. In R each factor variable has an associated set of linear contrasts, which are used as parametrization constraints in the fitting of the model specified in the `aov` function. These linear contrasts determine the estimated values of the unknown parameters. They can also be used to partition the associated sum of squares in the analysis of variance table using the `split` option to `summary(aov)`.

This dual use of contrasts for factor variables is very powerful, although somewhat confusing. We will first indicate the use of contrasts in estimation, before using them to partition the sums of squares.

The default contrasts for an unordered factor, which is created by `factor(x)`, are *treatment contrasts*, which are not strictly speaking contrasts as the columns don't sum to zero and are not orthogonal to the vector of ones. Treatment contrasts do, however, give a comparison of each treatment level relative to the first. This would be useful if, say, the first level were the control treatment. The default contrasts for an unordered factor in S-PLUS are *Helmert contrasts*, which compare the second level with the first, the third level with the average of the first two, and so on. Default contrasts for an ordered factor, in both S-PLUS and R, are those determined by the appropriate orthogonal polynomials. The contrasts used in fitting can be changed before an analysis of variance is constructed, using the `options` function, for example:

```
> options(contrasts = c("contr.sum",  "contr.poly"))
> options(contrasts = c("contr.helmert",  "contr.poly"))
```

imposes either the summation constraint $\Sigma\tau_j = 0$, or the Helmert constraints, respectively, for unordered factors, and orthogonal polynomial contrasts for ordered factors.

It is possible to specify a different set of contrasts for ordered factors from polynomial contrasts, but this will rarely be needed. In Section C.3.3 below we estimate the treatment parameters under each of the three constraints: Helmert, summation and $\tau_1 = 0$. If individual estimates of the τ_j are to be used for any purpose, and this should be avoided as far as feasible, it is essential to note the constraints under which these estimates were obtained.

The contrasts used in fitting the model can also be used to partition the sums of squares. The summation contrasts will rarely be of interest in this context, but the orthogonal polynomial contrasts will be useful for quantitative factors. Prespecified contrasts may also be specified, using the function `contrasts` or `C`. Use of the contrast matrix `C` is outlined in detail by Venables and Ripley (2002, Chapter 6.2).

C.2.5 Plotting

There are some associated plotting methods that are often useful. The function `interaction.plot` plots the mean response by levels of two cross-classified factors, and is illustrated in Section C.5 below. An optional argument `fun=` allows some other specified function of the response, such as the median or the standard error, to be plotted instead; see the help file for this function.

The function `qqnorm.aov()/qqnorm()` in package `gplots`, when applied to an analysis of variance object created by the `aov` function, constructs a full or half-normal plot of the estimated effects (see Section 5.5). Two optional arguments are very useful: `qqnorm.aov(aov.example, label = T)` allows interactive labeling of points in the plot by clicking on them, and `qqnorm(aov.example, full = T)` will construct a full normal plot of the estimated effects.

C.2.6 Specialized functions for standard designs

There are a number of functions for constructing designs in the packages `BHH2`, `DoE.base`, and `conf.design`; see the bibliographic notes. In the package `conf.design`, the function `conf.design` constructs symmetric confounded factorial designs. The package `BHH2` provides construction of fractional and full factorials for 2-level factors via `ffDesMatrix`. In the package `DoE.base`, `fac.design` and `oa.design`, are particularly useful for constructing design objects. Details on the use of these functions are given in the help files, as well as in the package manuals available through `cran.r-project.org`.

C.2.7 Missing values

Missing values are generally assigned the special value `NA`. `R` functions differ in their handling of missing values. Many of the plotting functions, for example, will plot missing values as zeroes; the documentation for, for example, `interaction.plot` includes under the description of the response variable the information "Missing values (NA) are allowed". On the other hand, `aov` handles missing values in the same way `lm` does, through the optional argument `na.action`. The default value for `na.action` is `na.omit`, which will omit any rows of the data frame that have missing values. An alternative is `na.fail`, which halts further computation.

In some design and analysis textbooks there are formulae for computing, by hand, treatment contrasts, standard errors, and analysis of variance tables in the presence of a small number of missing responses in randomized block designs; Cochran and Cox (1958) provide details for a number of other more complex designs. In general, procedures for arbitrarily unbalanced data may have to be used.

C.3 Randomized block experiment from Chapter 3

C.3.1 Data entry

This is the randomized block experiment taken from Cochran and Cox (1958), to compare five quantities of potash fertiliser on the strength of cotton fiber. The data and analysis of variance are given in Tables 3.1 and 3.2. The dependent variable is strength, and there are two classification variables, treatment (amount of potash), and block. The simplest way to enter the data is within R:

```
> potash.strength <- scan()
1: 762 814 776 717 746 800 815 773 757 768 793 787 774 780 721
16:
> potash.strength <- potash.strength/100
> potash.tmt <- factor(rep(1:5, 3))
> potash.blk <- factor(rep(1:3, each = 5))
> potash.tmt
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
 Levels: 1 2 3 4 5
> potash.blk
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
 Levels: 1 2 3
> is.factor(potash.tmt)
 [1] TRUE
```

We could also construct a 15×3 matrix to hold the response variable and the explanatory variables, although the columns of this matrix are all considered numeric, even if the variable entered is a factor.

```
> potash.matrix <- matrix(c(potash.strength, potash.tmt, potash.blk),
+ nrow = 15, ncol = 3)
```

```
> potash.matrix
      [, 1] [, 2] [, 3]
 [1, ] 7.62    1    1
 [2, ] 8.14    2    1
 [3, ] 7.76    3    1
 [4, ] 7.17    4    1
     .
     .
     .
 [15, ] 7.21   5    3
> is.factor(potash.tmt)
[1] TRUE
> is.factor(potash.matrix[, 2])
[1] FALSE
```

Finally we can construct the factor levels by by using `fac.design` in the package `DoE.base`, store the result in the design object `potash.design`, and combine this with the dependent variable in a data frame `potash.df`. In the illustration below we add names for the factor levels, an option that is available (but not required) in the `fac.design` function.

```
> library(DoE.base)
> fnames <- list (tmt=c("36","54","72","108","144"),
+                         blk=c("I","II","III") )
> potash.design <- fac.design(factor.names=fnames,
+  nlevels=c(5,3),randomize=F)
creating full factorial with  15  runs ...
> potash.design
   tmt blk
1   36   I
2   54   I
3   72   I
4  108   I
     .
     .
     .
15 144 III
class=design, type= full factorial
>
> strength<-potash.strength #use a shorter name
> potash.df <- data.frame(strength,potash.design)
> rm(strength,fnames,potash.design) # remove un-needed objects
> potash.df
   strength tmt blk
1      7.62  36   I
2      8.14  54   I
3      7.76  72   I
4      7.17 108   I
     .
     .
     .
15     7.21 144 III
```

```
> is.factor(potash.df$tmt)
[1] TRUE
> is.factor(potash.df$blk)
[1] TRUE
```

C.3.2 Table of treatment and block means

The simplest way to compute the treatment means is using the `tapply` function. When used with an optional factor argument as `tapply(y, factor, mean)` the calculation of the mean is stratified by the level of the factor. This can be used on any of the data structures outlined in the previous subsection:

```
> tapply(potash.strength, potash.tmt, mean)
   1      2      3      4    5
 7.85 8.0533 7.7433 7.5133 7.45

> tapply(potash.matrix[, 1], potash.matrix[, 2], mean)
   1      2      3      4    5
 7.85 8.0533 7.7433 7.5133 7.45

> tapply(potash.df$strength,  potash.df$tmt,  mean)
  36     54     72    108  144
 7.85 8.0533 7.7433 7.5133 7.45
```

As is apparent above, the `tapply` function is not terribly convenient when used on a data matrix or a data frame. There are special plotting methods for data frames with factors that allow easy plotting of the treatment means, but curiously there does not seem to be a ready way to print the treatment means without first constructing an analysis of variance.

C.3.3 Analysis of variance

We first form a two-way analysis of variance using `aov`. Note that the `summary` method for the analysis of variance object gives more useful output than printing the object itself.

In this example we illustrate the estimates $\hat{\tau}_j$ in the model $y_{js} = \mu + \tau_j + \beta_s + \epsilon_{js}$ under the default constraint specified by the treatment contrasts in R, with constraint $\tau_1 = 0$, which contrasts each level with the baseline level (specified by base), under the summation constraint $\sum \tau_j = 0$, and under the Helmert constraint which contrasts the second level with the first, the third with the average of the first two, and so on. If individual estimates of the τ_j are to be used for any purpose, it is essential to note the constraints under which these estimates were obtained. The analysis of variance table and estimated residual sum of squares are of course invariant to the choice of parametrization constraint.

```
> potash.aov <- aov(strength~tmt+blk, data = potash.df)
> potash.aov
Call:
   aov(formula  =  strength ~ tmt + blk,  data  =  potash.df)

Terms:
                    tmt     blk Residuals
Sum of Squares  0.73244 0.09712   0.34948
Deg. of Freedom       4       2         8

Residual standard error: 0.2090096 Estimated effects are balanced

> summary(potash.aov)
            Df  Sum Sq Mean Sq F value  Pr(>F)
tmt          4 0.73244 0.18311  4.1916 0.04037 *
blk          2 0.09712 0.04856  1.1116 0.37499
Residuals    8 0.34948 0.04369
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> coef(potash.aov) # same result as: potash.aov$coef

 (Intercept)   tmt54    tmt72   tmt108   tmt144  blkII  blkIII
      7.758   0.20333 -0.10667 -0.33667   -0.4  0.196    0.08

## default is contr.treatment; tmt36 and blkI set to 0

> options(contrasts = c("contr.sum", "contr.poly"))
> potash.aov <- aov(strength~tmt+blk,  data  =  potash.df)
> coef(potash.aov)
 (Intercept)  tmt1    tmt2     tmt3     tmt4     blk1    blk2
      7.722  0.128  0.33133  0.021333  -0.20867  -0.092  0.104

> options(contrasts = c("contr.helmert",  "contr.poly"))
> potash.aov <- aov(strength~tmt+blk,  data  =  potash.df)
> coef(potash.aov)  # same result as: potash.aov$coef
 (Intercept)    tmt1      tmt2      tmt3   tmt4  blk1   blk2
      7.722  0.10167 -0.069444 -0.092222 -0.068 0.098 -0.006
```

The estimated treatment effects under the summation constraint can also be obtained using `model.tables` or `dummy.coef`, so it is not necessary to change the default fitting constraint with the `options` function, although it is probably advisable. Below we illustrate this, assuming that the Helmert contrasts were used in the `aov` function. We also illustrate how `model.tables` can be used to obtain treatment means and their standard errors.

```
> options(contrasts = c("contr.helmert",  "contr.poly"))
> options("contrasts")
```

```
$contrasts [1] "contr.helmert" "contr.poly"

> dummy.coef(potash.aov)
Full coefficients are

(Intercept):          7.722
tmt:                     36         54         72        108        144
                      0.128      0.33133    0.021333    -0.20867     -0.272
blk:                      I          II         III
                     -0.092       0.104      -0.012

> model.tables(potash.aov)
Tables of effects

 tmt
tmt
     36      54      72     108     144
 0.1280  0.3313  0.0213 -0.2087 -0.2720

 blk
blk
     I     II    III
-0.092  0.104 -0.012

> model.tables(potash.aov, type = "means", se = T)
Tables of means Grand mean

7.722

 tmt
tmt
   36    54    72   108   144
7.850 8.053 7.743 7.513 7.450

 blk
blk
    I    II   III
7.630 7.826 7.710

Standard errors for differences of means
           tmt    blk
        0.1707 0.1322
replic.      3      5
```

C.3.4 Partitioning sums of squares

For the potash experiment, the treatment was a quantitative factor, and in Section 3.5.5 we discussed partitioning the treatment sums of squares using the linear and quadratic polynomial contrasts for a factor with five levels using $(-2, -1, 0, 1, 2)$ and $(2, -1, -2, -1, 2)$.

Since orthogonal polynomials are the default for an ordered factor, the simplest way to partition the sums of squares in R is to define `tmt` as an ordered factor.

```
> otmt <- ordered(potash.df$tmt)
> is.ordered(otmt)
[1] TRUE
> is.factor(otmt)
[1] TRUE
> contrasts(otmt)
              .L        .Q           .C        ^4
[1,] -6.32456e-01  0.534522 -3.16228e-01  0.119523
[2,] -3.16228e-01 -0.267261  6.32456e-01 -0.478091
[3,] -3.28798e-17 -0.534522  1.59520e-16  0.717137
[4,]  3.16228e-01 -0.267261 -6.32456e-01 -0.478091
[5,]  6.32456e-01  0.534522  3.16228e-01  0.119523

> potash.df <- data.frame(potash.df, otmt)
> rm(otmt)
> potash.aov <- aov(strength~otmt+blk, potash.df)
> summary(potash.aov)
            Df  Sum Sq Mean Sq F value  Pr(>F)
otmt         4 0.73244 0.18311  4.1916 0.04037 *
blk          2 0.09712 0.04856  1.1116 0.37499
Residuals    8 0.34948 0.04369
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> summary(potash.aov, split = list(otmt = list(L = 1, Q = 2)))
            Df  Sum Sq Mean Sq F value   Pr(>F)
otmt         4 0.73244 0.18311  4.1916 0.040368 *
  otmt: L    1 0.53868 0.53868 12.3310 0.007943 **
  otmt: Q    1 0.04404 0.04404  1.0081 0.344761
blk          2 0.09712 0.04856  1.1116 0.374985
Residuals    8 0.34948 0.04369
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> summary(potash.aov, split =
+                        list(otmt = list(L = 1, Q = 2, C = 3, QQ = 4)))
            Df  Sum Sq Mean Sq F value   Pr(>F)
otmt         4 0.73244 0.18311  4.1916 0.040368 *
  otmt: L    1 0.53868 0.53868 12.3310 0.007943 **
  otmt: Q    1 0.04404 0.04404  1.0081 0.344761
  otmt: C    1 0.13872 0.13872  3.1755 0.112609
  otmt: QQ   1 0.01100 0.01100  0.2518 0.629296
blk          2 0.09712 0.04856  1.1116 0.374985
Residuals    8 0.34948 0.04369
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

It is possible to specify just one contrast of interest, and a set of contrasts orthogonal to the first will be constructed automatically.

This set will not necessarily correspond to orthogonal polynomials however.

```
> contrasts(potash.tmt) <- c(-2, -1, 0, 1, 2)
> contrasts(potash.tmt)       #these contrasts are orthogonal
                              #but not the usual polynomial contrasts
  [, 1]     [, 2]    [, 3]    [, 4]
1   -2 -0.41491 -0.3626 -0.3104
2   -1  0.06722  0.3996  0.7320
3    0  0.83771 -0.2013 -0.2403
4    1 -0.21744  0.6543 -0.4739
5    2 -0.27258 -0.4900  0.2925
> potash.aov <- aov(potash.strength~potash.tmt+potash.blk)
> summary(potash.aov, split = list(potash.tmt = list(1)))
                 Df  Sum Sq Mean Sq F value   Pr(>F)
potash.tmt        4  0.7324  0.1831    4.19   0.0404 *
  potash.tmt: C1  1  0.5387  0.5387   12.33   0.0079 **
potash.blk        2  0.0971  0.0486    1.11   0.3750
Residuals         8  0.3495  0.0437
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

Finally, in this example recall that the treatment levels are not in fact equally spaced, so that the exact linear contrast is as given in Section 3.5: $(-2, -1.23, -0.46, 1.08, 2.6)$. This can be specified using `contrasts`, as illustrated here.

```
> contrasts(potash.tmt) <- c(-2, -1.23, -0.46, 1.08, 2.6)
> contrasts(potash.tmt)
  [, 1]     [, 2]    [, 3]    [, 4]
1 -2.00 -0.44375 -0.4103 -0.3773
2 -1.23 -0.09398  0.3332  0.7548
3 -0.46  0.86128 -0.1438 -0.1488
4  1.08 -0.15416  0.6917 -0.4605
5  2.60 -0.16939 -0.4707  0.2318

# as above these are not the usual orthogonal contrasts

> potash.aov <- aov(potash.strength~potash.tmt+potash.blk)
> summary(potash.aov, split = list(potash.tmt = list(1, 2, 3, 4)))
                 Df  Sum Sq Mean Sq F value   Pr(>F)
potash.tmt        4 0.73244 0.18311  4.1916 0.040368 *
  potash.tmt: C1  1 0.56677 0.56677 12.9740 0.006963 **
  potash.tmt: C2  1 0.00023 0.00023  0.0052 0.944440
  potash.tmt: C3  1 0.00445 0.00445  0.1019 0.757733
  potash.tmt: C4  1 0.16100 0.16100  3.6854 0.091153 .
potash.blk        2 0.09712 0.04856  1.1116 0.374985
Residuals         8 0.34948 0.04369
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

The function `poly()` will generate orthonormal polynomial contrasts for unequally spaced factor levels.

```
> tmtlev  <-  c(36, 54, 72, 108, 144)
>
> lincoef  <-  poly(tmtlev, degree = 1)
> lincoef
              1
[1, ] -0.5397956
[2, ] -0.3321819
[3, ] -0.1245682
[4, ]  0.2906592
[5, ]  0.7058866
      ...
> lincoef  <-  3.7*lincoef     #  scaling up
> print(lincoef, digits = 2)   #  round up to 2 digits for comparison with
                               #  result from previous "contrasts"
         1
[1, ] -2.00
[2, ] -1.23
[3, ] -0.46
[4, ]  1.08
[5, ]  2.61
      ...
```

C.4 Analysis of block designs in Chapter 4

C.4.1 Balanced incomplete block design

The first example in Section 4.2.6 is a balanced incomplete block design with two treatments per block in each of 15 blocks. The data are entered as follows:

```
> weight <- scan()
1: 251 215 249 223 254 226 258 215 265 241
11: 211 190 228 211 215 170 232 253 215 223
21: 234 215 230 249 220 218 226 243 228 256
31:
Read 30 items
> weight <- weight/100
> blk <- factor(rep(1:15, each = 2))
> blk
 [1] 1  1  2  2  3  3  4  4  ...15 15
> tmt  <-  0
> for (i in 1:5) for (j in (i+1):6) tmt  <-  c(tmt, i, j)
> tmt  <-  tmt[-1]
> tmt  <-  factor(tmt)
> tmt
 [1] 1 2 1 3 1 4 1 5 1 6 2 3 2 4 2 5 2 6 3 4 3 5 3 6 4 5 4 6 5 6
Levels: 1 2 3 4 5 6
> fnames <- c("C", "His-", "Arg-", "Thr-", "Val-", "Lys-")
> for (i in 1:6) levels(tmt)[i]  <-  fnames[i]
> rm(fnames)
> tmt <- factor(tmt)
> chick.df <- data.frame(weight, tmt, blk)
> chick.df
  weight  tmt blk
 1   2.51    C   1
 2   2.15 His-   1
 3   2.49    C   2
 4   2.23 Arg-   2
 5   2.54    C   3
 6   2.26 Thr-   3
 .
 .
 .
```

We now compute treatment means, both adjusted and unadjusted, and the analysis of variance table for their comparison. This is our first example of an unbalanced design, in which for example the sums of squares for treatments ignoring blocks is different from the sums of squares adjusted for blocks. The convention in R is that terms are added to the model in the order they are listed in the model statement. Thus to construct the intrablock analysis of variance, in which treatments are adjusted for blocks, we use the model statement `y ~ block + treatment`.

We used `tapply` to obtain the unadjusted treatment means, and obtained the adjusted means by adding $\hat{\tau}_j$ to the overall mean $\bar{Y}_{..}$. The $\hat{\tau}_j$ were obtained under the summation constraint. According to its help file, `model.tables (aov, type="means")` returns unadjusted means, but we do not recommend it; it seems to give incorrect results for the mean as well as for the standard error. The least squares estimates of τ_j under the summation constraint are returned by `dummy.coef`, even if the summation constraint option was not specified in fitting the model.

```
> tapply(weight, tmt, mean)
    C  His-  Arg-  Thr-  Val-  Lys-
2.554 2.202 2.184 2.212 2.092 2.484
> options(contrasts = c("contr.sum", "contr.poly"))
> chick.aov <- aov(weight~blk+tmt,  data  =  chick.df)
> summary(chick.aov)
           Df  Sum Sq Mean Sq F value   Pr(>F)
blk        14 0.75288 0.05378  8.1728 0.001025 **
tmt         5 0.44620 0.08924 13.5623
0.000347 *** Residuals   10 0.06580 0.00658
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> coef(chick.aov)
 (Intercept)    blk1   blk2   blk3     blk4     blk5       blk6
      2.288 -0.1105 -0.013 0.0245 0.060333  0.060333   -0.25883


     blk7    blk8   blk9      blk10  blk11  blk12  blk13  blk14
 -0.071333 -0.2705 0.0645 -0.0088333 0.117  0.102 0.0595 0.0495

   tmt1     tmt2      tmt3      tmt4     tmt5
 0.26167 0.043333 -0.091667 -0.086667 -0.22833

> dummy.coef(chick.aov)
Full coefficients are (Intercept):         2.288
 ...
tmt:       C     His-     Arg-     Thr-     Val-     Lys-
      0.26167  0.04333 -0.09167 -0.08667 -0.22833  0.10167

> tauhat  <- .Last.value$tmt   # same as tauhat <- dummy.coef(chick.aov)$tmt
> tauhat+mean(weight)  # adjusted mean
     C   His-   Arg-   Thr-   Val-   Lys-
 2.5497 2.3313 2.1963 2.2013 2.0597 2.3897

> model.tables(chick.aov, type = "means", se = T)
Tables of means Grand mean   2.288

...

tmt
    C  His-  Arg-  Thr-  Val-  Lys-
2.445 2.314 2.233 2.236 2.151 2.349

Standard errors for differences of means
           blk     tmt
        0.08112 0.05130
replic.       2       5
## these do not seem to be correctly adjusted for block effects
```

We will now compute the interblock analysis of variance using regression on the block totals. The most straightforward approach is to compute the estimates directly from equations (4.32) and (4.33); the estimated variance is obtained from the analysis of variance table with blocks adjusted for treatments. To obtain this analysis of variance table we specify treatment first in the right hand side of the model statement that is the argument of the `aov` function.

```
> N  <-  matrix(0,  nrow = 6,  ncol = 15)
> ind  <-  0
> for (i in 1:5) for (j in (i+1):6) ind  <-  c(ind, i, j)
> ind <-  ind[-1]
> ind  <-  matrix(ind,  ncol = 2, byrow = T)
> for (i in 1:15) N[ind[i, 1], i]  <-  N[ind[i, 2], i]  <- 1
> B <- tapply(weight, blk, sum)
> B
   1    2   3    4    5    6    7    8    9   10   11   12
 4.66 4.72 4.8 4.73 5.06 4.01 4.39 3.85 4.85 4.38 4.49 4.79

  13   14   15
 4.38 4.69 4.84
```

```
> tau  <- (N%*%B-5*2*mean(weight))/4
> tau  <- as.vector(tau)
> tau
[1]  0.2725 -0.2800 -0.1225 -0.0600 -0.1475  0.3375

> summary(aov(weight~tmt+blk,  data = chick.df))

            Df  Sum Sq Mean Sq F value    Pr(>F)
tmt          5 0.85788 0.17158 26.0754 1.963e-05 ***
blk         14 0.34120 0.02437
3.7039   0.02165 * Residuals   10 0.06580 0.00658
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> sigmasq <- 0.00658
> sigmaBsq <- ((0.34120/14-0.00658)*14)/(6*4)
> sigmaBsq
[1] 0.01037833
> vartau1  <-  sigmasq*2*5/(6*6)
> vartau2  <-  (2*5*(sigmasq+2*sigmaBsq))/(6*4)
> (1/vartau1)+(1/vartau2)
[1] 634.9066
> (1/vartau1)/.Last.value
[1] 0.8617211

> dummy.coef(chick.aov)$tmt
       C     His-     Arg-     Thr-     Val-     Lys-
 0.26167  0.04333 -0.09167 -0.08667 -0.22833  0.10167

> tauhat  <-  .Last.value
> taustar  <-  .86172*tauhat+(1-.86172)*tau

### these do not agree exactly with the text Table 4.12

> taustar
      C       His-     Arg-      Thr-     Val-    Lys-
 0.26316 -0.0013772 -0.09593 -0.082979 -0.21716 0.13428

> sqrt(1/((1/vartau1)+(1/vartau2)))
[1] 0.03968671
> setaustar <- .Last.value
> sqrt(2)*setaustar
[1] 0.05612548
```

C.4.2 Unbalanced incomplete block experiment

The second example from Section 4.2.6 has all treatment effects highly aliased with blocks. The data is given in Table 4.13 and the analysis summarized in Tables 4.14 and 4.15. The within block analysis is computed using the `aov` function, with blocks (days) entered into the model before treatments. The adjusted treatment means are computed by adding $\bar{Y}_{..}$ to the estimated coefficients. We also indicate the computation of the least squares estimates under the summation constraint using the matrix formulae of Section 4.2. The contrasts between pairs of treatment means do not have equal precision; the estimated standard error is computed for each mean using $\text{var}(\bar{Y}_{j.}) = \sigma^2/r_j$, although for comparing pairs of means it may be more useful to use the result that $\text{cov}(\hat{\tau}) = C^-$.

```
> day <- rep(1:7, each = 4)
> tmt <- scan()
1: 1 8 9 9 9 5 4 9 2 3 8 5 12 6 14 10
17: 11 15 3 13 1 6 4 7 2 9 7 9
29:
Read 28 items
> expansion  <-  scan()
1: 150 148 130 117 122 141 112 116
9: 159 108 158 156 127 186 114 112
17: 130 111 101 117 146 178 128 154
25: 150 107 109 96
29:
Read 28 items

> day <- factor(day)
> tmt <- factor(tmt)
> expansion <- expansion/10
> dough.df  <-  data.frame(expansion, tmt, day)
> dough.df
  expansion tmt day
 1      15.0   1   1
 2      14.8   8   1
 3      13.0   9   1
 4      11.7   9   1
 5      12.2   9   2
 6      14.1   5   2
        .
        .
        .

> tapply(expansion, day, mean)
     1      2      3      4      5      6      7
13.625 12.275 14.525 13.475 11.475 15.150 11.550

> tapply(expansion, tmt, mean)
   1     2     3  4     5    6     7    8        9   10 11
 14.8 15.45 10.45 12 14.85 18.2 13.15 15.3 11.46667 11.2 13

  12   13   14   15
 12.7 11.7 11.4 11.1

> options(contrasts = c("contr.helmert",  "contr.poly"))
> dough.aov  <-  aov(expansion~day+tmt,  data = dough.df)
> summary(dough.aov)
           Df Sum Sq Mean Sq F value   Pr(>F)
day         6 49.412   8.235 11.1877 0.002750 **
tmt        14 96.225   6.873  9.3372 0.003149 **
Residuals   7  5.153   0.736
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> dummy.coef(.Last.value)$tmt
     1      2       3       4      5      6       7      8
 1.3706 3.5372 -2.3156 -1.0711 2.1622 3.9178 0.85389 2.2539

       9      10      11      12       13      14      15
 -0.51556 -3.4822 0.58444 -1.9822 -0.71556 -3.2822 -1.3156

> replications(dough.df)
$expansion NULL

$tmt
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
 2  2  2  2  2  2  2  2  6  1  1  1  1  1  1

$day
[1] 4

> R <- matrix(0, nrow = 15, ncol = 15)
> diag(R)  <-  replications(dough.df)$tmt
> K  <-  matrix(0, nrow = 7, ncol = 7)
> diag(K)  <-  4
> N <- matrix(0, nrow = 15, ncol = 7)
> N[, 1] <- c(1, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0)
> N[, 2] <- c(0, 0, 0, 1, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0)
> N[, 3] <- c(0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
> N[, 4] <- c(0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0)
> N[, 5] <- c(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1)
> N[, 6] <- c(1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)
> N[, 7] <- c(0, 1, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0, 0)
>
> S <- tapply(expansion, tmt, sum)
> S <- matrix(S)
>
> B <- tapply(expansion, day, sum)
> B <- matrix(B)
>
> library(MASS)
> Q <- S-N%*%solve(K)%*%B
> C <- R-N%*%solve(K)%*%t(N)
> t(Q)%*%ginv(C)
       [, 1]   [, 2]    [, 3]    [, 4]   [, 5]   [, 6]    [, 7]   [, 8]
[1, ] 1.3706 3.5372 -2.3156 -1.0711 2.1622 3.9178 0.85389 2.2539

         [, 9]   [, 10]   [, 11]   [, 12]    [, 13]   [, 14]   [, 15]
[1, ] -0.51556 -3.4822 0.58444 -1.9822 -0.71556 -3.2822 -1.3156
```

```
> tauhat <- .Last.value
> as.vector(tauhat+mean(expansion))
 [1] 14.5241 16.6908 10.8380 12.0825 15.3158 17.0713 14.0075
 [8] 15.4075 12.6380  9.6713 13.7380 11.1713 12.4380  9.8713
[15] 11.8380
> se <- 0.7361/sqrt(diag(R))
> se
 [1] 0.52050 0.52050 0.52050 0.52050 0.52050 0.52050 0.52050
 [8] 0.52050 0.30051 0.73610 0.73610 0.73610 0.73610 0.73610
[15] 0.73610
> setauhat <- sqrt(diag(ginv(C)))
> setauhat
 [1] 0.92376 0.92376 1.04243 0.92376 0.92376 1.04243 0.92376
 [8] 0.92376 0.76594 1.59792 1.59792 1.59792 1.59792 1.59792
[15] 1.59792
```

C.5 Examples from Chapter 5

C.5.1 Factorial experiment, Section 5.2

The treatments in this experiment form a complete $3 \times 2 \times 2$ factorial. The data are given in Table 5.1 and the analysis summarized in Tables 5.2 and 5.4. The code below illustrates how to construct the levels of the factors. For this purpose we treat house as a factor, although in line with the discussion of Section 5.1 it is not an aspect of treatment. These factors are then used to stratify the response in the `tapply` function, producing tables of marginal means. Figure 5.1 was obtained using `interaction.plot`, after constructing a four-level factor indexing the four combinations of type of protein crossed with level of fish solubles.

```
> weight <- scan()
1: 6559 6292 7075 6779 6564 6622 7528 6856 6738 6444 7333 6361
13: 7094 7053 8005 7657 6943 6249 7359 7292 6748 6422 6764 6560
25:
Read 24 items
> library(DoE.base)
> fnames <- list (House=c("I","II"), Lev.f=c("0","1"),
+                 Lev.pro=c("0","1","2"), Type=c("gnut","soy") )

> exk.design <-fac.design(factor.names=fnames,
+                        nlevels=c(2,2,3,2),randomize=F)
creating full factorial with  24  runs ...
> exk.design
   House Lev.f Lev.pro Type
1      I     0       0 gnut
2     II     0       0 gnut
3      I     1       0 gnut
4     II     1       0 gnut
```

```
5      I     0       1 gnut
        .
        .
        .

class=design, type= full factorial
> exk.df <- data.frame(weight, exk.design)
> rm(exk.design)
> exk.df
   weight House Lev.f Lev.pro Type
1    6559     I     0       0 gnut
2    6292    II     0       0 gnut
3    7075     I     1       0 gnut
4    6779    II     1       0 gnut
        .
        .
        .
24   6560    II     1       2  soy

> tapply(weight, list(exk.df$Lev.pro, exk.df$Type), mean)
     gnut     soy
0 6676.25 7452.25
1 6892.50 6960.75
2 6719.00 6623.50
> tapply(weight, list(exk.df$Lev.f, exk.df$Type), mean)
      gnut      soy
0 6536.500 6751.500
1 6988.667 7272.833
> tapply(weight, list(exk.df$Lev.f, exk.df$Lev.pro), mean)
       0       1      2
0 6749.5 6594.50 6588.0
1 7379.0 7258.75 6754.5
> tapply(weight, list(exk.df$Lev.pro, exk.df$Lev.f, exk.df$Type),
              mean)

, , gnut
       0    1
0 6425.5 6927
1 6593.0 7192
2 6591.0 6847

, , soy
       0      1
0 7073.5 7831.0
1 6596.0 7325.5
2 6585.0 6662.0

> Type.Lev.f <- factor(c(1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2,
+ 3, 3, 4, 4, 3, 3, 4, 4, 3, 3, 4, 4))
> postscript(file = "Fig5.1.ps", horizontal = F)
> interaction.plot(exk.df$Lev.pro, Type.Lev.f, weight,
+          xlab="Level of Protein")

## the legend in Chapter 5 is incorrect; see the Errata
```

```
> dev.off()
```

Table 5.3 shows the analysis of variance, using interactions with houses as the estimate of error variance. As usual, the summary table for the analysis of variance includes calculation of F statistics and associated p-values, whether or not these make sense in light of the design. For example, the F statistic for the main effect of houses does not have a justification under the randomization, which was limited to the assignment of chicks to treatments. Individual assessment of main effects and interactions via F-tests is also usually not relevant; the main interest is in comparing treatment means. As the design is fully balanced, `model.tables` provides a set of cross-classified means, as well as the standard errors for their comparison. The linear and quadratic contrasts for the three-level factor level of protein are obtained first by defining protein as an ordered factor, and then by using the `split` option to the analysis of variance summary.

```
> exk.aov <- aov(weight~Lev.f*Lev.pro*Type+House,  data = exk.df)
> summary(exk.aov)
                   Df  Sum Sq Mean Sq F value    Pr(>F)
Lev.f               1 1421553 1421553 31.7414 0.0001524 ***
Lev.pro             2  636283  318141  7.1037 0.0104535 *
Type                1  373751  373751  8.3454 0.0147366 *
House               1  708297  708297 15.8153 0.0021705 **
Lev.f:Lev.pro       2  308888  154444  3.4485 0.0687641 .
Lev.f:Type          1    7176    7176  0.1602 0.6966078
Lev.pro:Type        2  858158  429079  9.5808 0.0038964 **
Lev.f:Lev.pro:Type  2   50128   25064  0.5596 0.5868633
Residuals          11  492640   44785
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> model.tables(exk.aov, type = "mean", se = T)

Tables of means Grand mean

6887.375

 Lev.f
Lev.f
   0    1
6644 7131

. . .
Standard errors for differences of means
        Lev.f Lev.pro  Type House Lev.f:Lev.pro Lev.f:Type
         86.4   105.8  86.4  86.4         149.6      122.2
replic.    12       8    12    12             4          6
```

```
       Lev.pro:Type Lev.f:Lev.pro:Type
              149.6              211.6
replic.           4                  2

> options(contrasts = c("contr.poly", "contr.poly"))
> exk.aov2 <- aov(weight~Lev.f*Lev.pro*Type+House,  data = exk.df)
> summary(exk.aov2, split = list(Lev.pro = list(1, 2)))

                         Df  Sum Sq Mean Sq F value    Pr(>F)
Lev.f                     1 1421553 1421553 31.7414 0.0001524 ***
Lev.pro                   2  636283  318141  7.1037 0.0104535 *
  Lev.pro: C1             1  617796  617796 13.7946 0.0034167 **
  Lev.pro: C2             1   18487   18487  0.4128 0.5337216
Type                      1  373751  373751  8.3454 0.0147366 *
House                     1  708297  708297 15.8153 0.0021705 **
Lev.f:Lev.pro             2  308888  154444  3.4485 0.0687641 .
  Lev.f:Lev.pro: C1       1  214369  214369  4.7866 0.0511622 .
  Lev.f:Lev.pro: C2       1   94519   94519  2.1105 0.1742169
Lev.f:Type                1    7176    7176  0.1602 0.6966078
Lev.pro:Type              2  858158  429079  9.5808 0.0038964 **
  Lev.pro:Type: C1        1  759512  759512 16.9589 0.0017061 **
  Lev.pro:Type: C2        1   98645   98645  2.2026 0.1658565
Lev.f:Lev.pro:Type        2   50128   25064  0.5596 0.5868633
  Lev.f:Lev.pro:Type: C1  1   47306   47306  1.0563 0.3261338
  Lev.f:Lev.pro:Type: C2  1    2821    2821  0.0630 0.8064476
Residuals                11  492640   44785
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

C.5.2 2^{4-1} *fractional factorial; Section 5.7*

The data for the nutrition trial of Blot et al. (1993) is given in Table 5.9. Below we illustrate the analysis of the log of the death rate from cancer, and the numbers of cancer deaths. The second analysis is a reasonable approximation to the first as the numbers at risk are nearly equal across treatment groups. Both these analyses ignore the blocking information on sex, age and commune. Blot et al. (1993) report the results in terms of the relative risk, adjusting for the blocking factors; the conclusions are broadly similar. Here we illustrate the `oa.design` function in the package `DoE.base` to generate the design matrix. In the model formula the shorthand `.^ 2` denotes all main effects and two-factor interactions.

We illustrate the use of `qqnorm.aov` in the package `gplots` for constructing a half-normal plot of the estimated effects from an `aov` object. The function `qqnorm.aov(aov.object, full=T)` will produce a full-normal plot of the estimated effects, and effects other than the grand mean can be omitted from the plot with the option `omit=`. The effects are extracted from the `aov` object using `effects(aov-example)`, which in turn relies on the Q-R decomposition; these are not equal to, but are proportional to, the effects as defined as the average difference between the two levels.

```
> library(DoE.base) # for oa.design
> library(gplots)   # for qqnorm.aov

> lohi <- c("0","1")
> fnames <- list(D=lohi,C=lohi,B=lohi,A=lohi)
> d <-oa.design(factor.names=fnames,nruns=8, nfactors=4,
+                    nlevels=2,randomize=F)
> cancer.design <-cbind(d[,4],d[,3],d[,2],d[,1])
> cancer.design
  A B C D
1 0 0 0 0
2 1 1 0 0
3 1 0 1 0
        .
        .
        .
> death.c <- scan()
1: 107 94 121 101 81 103 90 95
9:
> mean(1/death.c)
[1] 0.01023017
> years <- scan()
1: 18626 18736 18701 18686 18745 18729 18758 18792
9:
> log.rates <- log(death.c/years)

# Below we analyse number of deaths from cancer and
# the log death rate; the latter is discussed in Section 5.7.

> logcancer.df<-data.frame(log.rates,cancer.design)
> logcancer.df
  log.rates  A  B  C  D
1 -5.159485 -1 -1 -1 -1
2 -5.294907  1  1 -1 -1
3 -5.040542  1 -1  1 -1
4 -5.220409 -1  1  1 -1
5 -5.444233  1 -1 -1  1
6 -5.203099 -1  1 -1  1
7 -5.339566 -1 -1  1  1
8 -5.287310  1  1  1  1

> cancer.df <- data.frame(death.c,  cancer.design)
> rm(lohi,death.c,log.rates,d,cancer.design)

> logcancer.aov <- aov(log.rates~.^2,  data = logcancer.df)
> model.tables(logcancer.aov,  type = "effects")
```

```
Tables of effects
A
       -1         1
 0.018054 -0.018054

B
        -1          1
 0.0027375 -0.0027375

C
       -1         1
-0.026737  0.026737

D
      -1        1
 0.06986 -0.06986

A:B
    B
A    -1        1
  -1 -0.021623  0.021623
  1   0.021623 -0.021623

A:C
    C
A    -1       1
  -1  0.07609 -0.07609
  1  -0.07609  0.07609

A:D
    D
A    -1        1
  -1 -0.029165  0.029165
  1   0.029165 -0.029165

> cancer.aov <- aov(death.c~.^2, data = cancer.df)
> model.tables(cancer.aov, type = "effects")

Tables of effects
A
   -1     1
 1.25 -1.25

B
   -1     1
 0.75 -0.75

C
   -1     1
-2.75  2.75

D
   -1     1
 6.75 -6.75
```

```
A:B
    B
A    -1   1
  -1 -2.5  2.5
  1   2.5 -2.5

A:C
    C
A    -1   1
  -1  7.5 -7.5
  1  -7.5  7.5

A:D
    D
A    -1 1
  -1 -3  3
  1   3 -3
> qqnorm(logcancer.aov, label = TRUE) # plot half-normal quantitle
                                      # then save it as FigC.1.ps
> mean(1/death.c)
[1] 0.01023017
```

C.5.3 Exercise 5.5: flour milling

This example is adapted from Tuck, Lewis and Cottrell (1993); that article provides a detailed case study of the use of response surface methods in a quality improvement study in the flour milling industry. A subset of the full data from the article's experiment I is given in Table 5.11. There are six factors of interest, all quantitative, labelled A through F and coded -1 and 1. The experiment forms a one-quarter fraction of a 2^6 factorial. The complete data included a further 13 runs taken at coded values for the factors arranged in what is called in response surface methodology a central composite design. Below we construct the fractional factorial by specifying the defining relations as an optional argument to `ffDesMatrix`.

```
> library(BHH2)
> M <- ffDesMatrix(6, gen = list(c(5, 1, 2, 3), c(6, 2, 3, 4)))

## A 2^(6-2) factorial design, with
## the alias structure  5=123 and 6=234;
## we label these acccording to Table 5.11

> A <- factor(M[, 4])
> B <- factor(M[, 3])
> C <- factor(M[, 2])
> D <- factor(M[, 6])
```

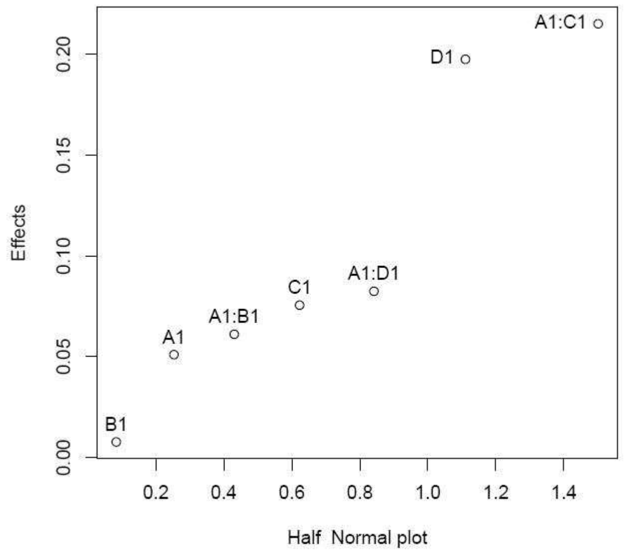

Figure C.1 *Half normal plots of estimated effects: cancer mortality in Linxiang nutrition trial. Aliased effects are automatically omitted. These "estimated effects" are proportional to, but not equal to, effects as defined in the text.*

```
> E <- factor(M[, 1])
> F <- factor(M[, 5])
```

```
> flour.y <- scan()
1: 519 446 337 415 503 468 343 418 567 471 355 424
13: 552 489 361 425 534 466 356 431 549 461 354 427
25: 560 480 345 437 535 477 363 418 558 483 376 418
37: 551 472 349 426 576 487 358 434 569 494 357 444
49: 562 474 358 404 569 494 348 400 568 478 367 463
61: 551 500 373 462
65:
> flour.tmt <- rep(1:16, each = 4)
> flour.tmt
 [1]  1  1  1  1  2  2  2  2  3  3  3  3  ...

> flour.tmt <- factor(flour.tmt)
> flour.day <- rep(1:4, 16)
> flour.day <- factor(flour.day)
> tapply(flour.y, flour.tmt, mean)
     1      2      3      4      5      6      7      8      9
429.25 433.00 454.25 456.75 446.75 447.75 455.50 448.25 458.75
```

```
    10     11     12     13     14     15     16
449.50 463.75 466.00 449.50 452.75 469.00 471.50
> flour.ybar <- .Last.value

> flour.df <- data.frame(flour.ybar, A, B, C, D, E, F)
> flour.df
   flour.ybar  A  B  C  D  E  F
1      429.25 -1 -1 -1 -1 -1 -1
2      433.00 -1 -1 -1 -1  1  1
3      454.25 -1 -1  1  1 -1  1
4      456.75 -1 -1  1  1  1 -1
5      446.75 -1  1 -1  1 -1  1
6      447.75 -1  1 -1  1  1 -1
7      455.50 -1  1  1 -1 -1 -1
8      448.25 -1  1  1 -1  1  1
9      458.75  1 -1 -1  1 -1 -1
10     449.50  1 -1 -1  1  1  1
11     463.75  1 -1  1 -1 -1  1
12     466.00  1 -1  1 -1  1 -1
13     449.50  1  1 -1 -1 -1  1
14     452.75  1  1 -1 -1  1 -1
15     469.00  1  1  1  1 -1 -1
16     471.50  1  1  1  1  1  1

> flour.aov <- aov(flour.ybar ~ A*B*C*D*E*F, data = flour.df)
> summary(flour.aov)
            Df   Sum Sq  Mean Sq
A            1   745.97   745.97
B            1    55.32    55.32
C            1   866.57   866.57
D            1   197.75   197.75
E            1     0.10     0.10
F            1    23.16    23.16
A:B          1    25.63    25.63
A:C          1     0.19     0.19
B:C          1    32.35    32.35
A:E          1     0.10     0.10
B:E          1 0.003906 0.003906
C:E          1     0.10     0.10
D:E          1     1.72     1.72
A:B:E        1    39.85    39.85
A:C:E        1    25.63    25.63

> flour.aov2 <- aov(flour.y ~ flour.tmt + flour.day)
> summary(flour.aov2)
            Df Sum Sq Mean Sq  F value    Pr(>F)
flour.tmt   15   8058     537   3.4284 0.0006867 ***
flour.day    3 324508  108169 690.3488 < 2.2e-16 ***
Residuals   45   7051     157
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> model.tables(flour.aov, type = "effects")
```

```
Tables of effects

A
    -1      1
-6.828  6.828

B
     -1       1
-1.8594  1.8594

C
    -1      1
-7.359  7.359

D
    -1      1
-3.516  3.516

E
      -1        1
 0.07812 -0.07812

F
     -1       1
 1.2031 -1.2031

A:B
    B
A    -1      1
  -1 -1.2656  1.2656
  1   1.2656 -1.2656

A:C
    C
A    -1       1
  -1  0.10937 -0.10937
  1  -0.10937  0.10937

B:C
    C
B    -1      1
  -1 -1.4219  1.4219
  1   1.4219 -1.4219

A:E
    E
A    -1       1
  -1 -0.07813  0.07813
  1   0.07813 -0.07813

B:E
    E
B    -1        1
  -1  0.015625 -0.015625
  1  -0.015625  0.015625

C:E
    E
C    -1       1
  -1  0.07812 -0.07812
  1  -0.07812  0.07812

D:E
    E
D    -1      1
  -1 -0.3281  0.3281
  1   0.3281 -0.3281

A:B:E

,  ,  E  =  -1
    B
A    -1      1
  -1 -1.5781  1.5781
  1   1.5781 -1.5781

,  ,  E  =  1
    B
A    -1      1
  -1  1.5781 -1.5781
  1  -1.5781  1.5781


 A:C:E

,  ,  E  =  -1
    C
A    -1      1
  -1 -1.2656  1.2656
  1   1.2656 -1.2656

,  ,  E  =  1
    C
A    -1      1
  -1  1.2656 -1.2656
  1  -1.2656  1.2656
```

C.6 Examples from Chapter 6

C.6.1 Split unit

The data for a split unit experiment are given in Table 6.9. The structure of this example is identical to the split unit example involving varieties of oats, originally given by Yates (1935), used as an illustration by Venables and Ripley (2002, Chapter 6.7). Their discussion of split unit experiments emphasizes their formal similarity to designs with more than one component of variance, such as discussed briefly in Section 6.5. From this point of view the subunits are nested within the whole units, and there is a special modelling operator `A/B` to represent factor B nested within factor A. Thus the result of

```
aov(y ~ temp * prep + Error(reps/prep))
```

is a list of `aov` objects, one of which is the whole unit analysis of variance and another is the subunit analysis of variance. The subunit analysis is implied by the model formula because the finest level analysis, in our case "within reps", is automatically computed. As with unbalanced data, `model.tables` cannot be used to obtain estimated standard errors, although it will work if the model statement is changed to omit the interaction term between preparation and temperature. Venables and Ripley (2002, Chapter 6.7) discuss the calculation of residuals and fitted values in models with more than one source of variation.

```
> y <- scan()
1: 30 34 29 35 41 26 37 38 33 36 42 36
13: 28 31 31 32 36 30 40 42 32 41 40 40
25: 31 35 32 37 40 34 41 39 39 40 44 45
37:
Read 36 items
> prep <- factor(rep(1:3, times = 12))
> temp <- factor(rep(rep(1:4, each = 3), times = 3))
> days <- factor(rep(1:3, each = 12))
>
> split.df <- data.frame(days, temp, prep, y)
> rm(y,  prep,  temp,  days)
> split.df
   days temp prep  y
1     1    1    1 30
2     1    1    2 34
3     1    1    3 29
4     1    2    1 35
     ...

> split.aov <- aov(y~temp*prep+Error(days/prep), data = split.df)
> summary(split.aov)

Error: days
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  2 77.556  38.778

Error: days:prep
          Df  Sum Sq Mean Sq F value  Pr(>F)
prep       2 128.389  64.194  7.0781 0.04854 *
```

```
Residuals  4  36.278   9.069
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Error: Within
          Df Sum Sq Mean Sq F value    Pr(>F)
temp       3 434.08  144.69 36.4266 7.449e-08 ***
temp:prep  6  75.17   12.53  3.1538   0.02711 *
Residuals 18  71.50    3.97
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> model.tables(split.aov, type = "mean")

Tables of means

Grand mean 36.02778

temp
    1     2     3     4
31.22 34.56 37.89 40.44

prep
    1     2     3
35.67 38.50 33.92

temp:prep
    prep
temp 1     2     3
   1 29.67 33.33 30.67
   2 34.67 39.00 30.00
   3 39.33 39.67 34.67
   4 39.00 42.00 40.33

# calculate errors by hand

# use whole plot error for prep;
# prep means are averaged over 12 observations
> sqrt(2*9.069/12)
[1] 1.229431

# use subplot error for temp;
# temp means are averaged over 9 observations
> sqrt(2*3.97/9) observations[1] 0.9392669

# use subplot error for temp:prep;
# these means are averaged over 3 observations
> sqrt(2*3.97/3)
[1] 1.626858
```

C.6.2 Wafer experiment; Section 6.7.2

There are six controllable factors and one noise factor. The design is a split plot with the noise factor, over-etch time, the sub plot treatment. Each subplot is an orthogonal array of 18 runs with six factors each at three levels. Tables of such arrays are available from the function `oa.design`.

The F-value and p-value have been deleted from the output, as the main effects of the factors should be compared using the whole plot error, and the interactions of the factors with OE should be compared using the subplot error. These two error components are not provided using the split plot formula, as there is no replication of the whole plot treatment. One way to extract them is to specify the model with all estimable interactions, and pool the appropriate (higher order) ones to give an estimate of the residual mean square.

```
> library(DoE.base)
> elect1 <- oa.design(L18, randomize=F, columns=c(2:7))
> elect1
   A B C D E F
1  1 1 1 1 1 1
2  1 2 2 2 2 2
3  1 3 3 3 3 3
4  2 1 1 2 2 3
5  2 2 2 3 3 1
      ...
18 3 3 2 1 2 3
class=design, type= oa

> OE <- factor(rep(c(1,2), each=18))
> elect.design <- cbind(elect1, OE)
Warning message:
In data.frame(..., check.names = FALSE) :
  row names were found from a short variable and have been discarded
> elect.design
   A B C D E F OE
1  1 1 1 1 1 1  1
2  1 2 2 2 2 2  1
3  1 3 3 3 3 3  1
      ...
35 3 2 1 3 1 2  2
36 3 3 2 1 2 3  2

> y <- scan()
1: 4750 5444 5802 6088 9000 5236 12960 5306 9370 4942
11: 5516 5084 4890 8334 10750 12508 5762 8692 5050 5884
21: 6152 6216 9390 5902 12660 5476 9812 5206 5614 5322
31: 5108 8744 10750 11778 6286 8920
37:
Read 36 items
>
```

```
> elect.df <- data.frame(y,elect.design)
> rm(y,elect1,elect.design,OE)
> elect.df
       y A B C D E F OE
1   4750 1 1 1 1 1 1  1
2   5444 1 2 2 2 2 2  1
3   5802 1 3 3 3 3 3  1
4   6088 2 1 1 2 2 3  1
  . . .
35  6286 3 2 1 3 1 2  2
36  8920 3 3 2 1 2 3  2

> rm(y, A, B, C, D, E, F, OE)

> elect.aov <- aov(y~(A+B+C+D+E+F)+OE+OE*(A+B+C+D+E+F), data = elect.df)
> summary(elect.aov)
            Df   Sum Sq  Mean Sq
A            2 84082743 42041371
B            2  6996828  3498414
C            2  3289867  1644933
D            2  5435943  2717971
E            2 98895324 49447662
F            2 28374240 14187120
OE           1   408747   408747
A:OE         2   112170    56085
B:OE         2   245020   122510
C:OE         2     5983     2991
D:OE         2   159042    79521
E:OE         2   272092   136046
F:OE         2    13270     6635
Residuals   10  4461690   446169
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> summary(elect.aov,split=list( A=list(1,2), B=list(1,2),C=list(1,2),
+                               D=list(1,2), E=list(1,2),F=list(1,2)))
            Df   Sum Sq  Mean Sq
A            2 84082743 42041371
  A: C1      1   590422   590422
  A: C2      1 83492321 83492321
B            2  6996828  3498414
  B: C1      1  6991307  6991307
  B: C2      1     5521     5521
C            2  3289867  1644933
  C: C1      1  3275947  3275947
  C: C2      1    13920    13920
D            2  5435943  2717971
  D: C1      1   702903   702903
  D: C2      1  4733040  4733040
E            2 98895324 49447662
  E: C1      1    42438    42438
  E: C2      1 98852886 98852886
F            2 28374240 14187120
  F: C1      1  1572947  1572947
  F: C2      1 26801293 26801293
OE           1   408747   408747
A:OE         2   112170    56085
  A:OE: C1   1    35556    35556
  A:OE: C2   1    76614    76614
B:OE         2   245020   122510
  B:OE: C1   1    70939    70939
  B:OE: C2   1   174081   174081
C:OE         2     5983     2991
  C:OE: C1   1      523      523
  C:OE: C2   1     5460     5460
D:OE         2   159042    79521
  D:OE: C1   1   133300   133300
  D:OE: C2   1    25741    25741
E:OE         2   272092   136046
  E:OE: C1   1    50139    50139
  E:OE: C2   1   221953   221953
F:OE         2    13270     6635
  F:OE: C1   1    12429    12429
  F:OE: C2   1      840      840
Residuals   10  4461690   446169
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> summary(aov(y~A*B*C*D*E*F*OE, data = elect.df))
            Df   Sum Sq  Mean Sq
A            2 84082743 42041371
B            2  6996828  3498414
C            2  3289867  1644933
D            2  5435943  2717971
E            2 98895324 49447662
F            2 28374240 14187120
OE           1   408747   408747
A:B          2   229714   114857
B:C          2  3001526  1500763
B:E          1  1175056  1175056
A:OE         2   112170    56085
B:OE         2   245020   122510
C:OE         2     5983     2991
D:OE         2   159042    79521
E:OE         2   272092   136046
F:OE         2    13270     6635
A:B:OE       2     2616     1308
B:C:OE       2    49258    24629
B:E:OE       1     3520     3520

> (229714+3001526+1175056)/5  # (AB+BC+BE)/5
[1] 881259.2
> (2616+49258+3520)/5         # (A:B:OE+B:C:OE+B:E:OE)/5
[1] 11078.8
```

C.7 Bibliographic notes

The definitive guide to statistical analysis with S-PLUS/R is Venables and Ripley (2002), now in its fourth edition. See also the book web page

`http://www.stats.ox.ac.uk/pub/MASS4/`

A detailed discussion of contrasts for fitting and partitioning sums of squares is given in Chapter 6.2, and analysis of structured designs is outlined in Chapter 6.7 and 6.8. Models with several components of variation are discussed in Chapter 6.11 and current releases of R include the `nlme` package for fitting mixed effects models.

The R web site
`http://cran.r-project.org/web/views/ExperimentalDesign.html`
lists a number of packages for experimental design and analysis of data from designed experiments with a wealth of related useful links.

Faraway's[||] *Practical Regression and Anova using R* (2002), gives a readable introduction to R with examples of the analysis of structured designs in Chapter 16. Another helpful reference is *An R companion to "Experimental Design"* by Vikneswaran,[**] a companion to Berger & Maurer.[††]

[||] `cran.r-project.org/doc/contrib/Faraway-PRA.pdf`

[**] `http://cran.r-project.org/doc/contrib/Vikneswaran-ED_companion.pdf`

[††] Berger, P.D. and Maurer, E. (2002). *Experimental Design with Applications in Management, Engineering and the Sciences*, Duxbury Press, Belmont.