An R companion to "Experimental Design"

Vikneswaran

Preface

This document is, as the title suggests, an R companion to the following book: *"Experimental Design with Applications in Management, Engineering and the Sciences"*, written by two professors who were with Boston University at the time. These are the full details of the book:

Title	Experimental Design with Applications in Management, Engineering and the Sciences
Authors	Paul D Berger & Robert E Maurer
Publisher	Duxbury
Publication Date	2002
ISBN	0-534-35822-5

It is informative to point out my purpose in writing this companion, because as Agent Smith so rightly pointed out,

> "... without purpose, we would not exist."

In essence, this document is a step-by-step guide on how to use R to carry out the analyses and techniques covered in the above book, which basically covers ANOVA techniques. As such, the chapters here mirror the books' (up to Chapter 11) and the datsets used are from the examples in the book. This companion might prove useful to students who are using the eponymous book as their textbook, and are just starting out in R.

It must be categorically stated that I do not wish to make any profit out of this document. I doubt if people would even consider paying for it anyway! The sole aim is to get more people to use and contribute to R, and to spread the love of statistics. I am currently not affiliated to any organisation or university, but can be reached at the email address below. Datsets used here can be downloaded from the website specified below.

Vikneswaran
viknesh_g@yahoo.co.uk
http://www.geocities.com/viknesh_g/
August 2005

Contents

List of Figures

Part I

Primary Focus on Factors Under Study

Chapter 1

Introduction to Experimental Design

Chapter 1 in the book, hereafter referred to as ED in order to save typing, is quite qualitative. Among other things, it states that there are six steps to experimental design, starting with planning the experiment and ending with evaluating the conclusions of the experiment. The chapter then goes on to discuss some real-world applications of designed experiments. R cannot really help much here.

Chapter 2

One-Factor Designs and the Analysis of Variance

This chapter in ED considers the impact of a single factor on some performance measure. In this document, the first section lays down some terminology and notations used, hopefully minimising confusion when ED delves into two and then three factor designs later. After that we cover the easiest way to read datasets into R. The third section is the first real meat: the `aov()` function. For the purpose of instruction, we use the battery-life example from ED.

2.1 Notation and Terminology

The variable Y will be used to denote the **dependent variable**. In various instances, it will also be referred to as the *yield, response variable* and *performance measure*. The **independent variable** will be X. The statistical model we desire to fit to the data is

$$Y_{ij} = \mu + \tau_j + \epsilon_{ij}$$

where

$$\begin{aligned}
i &= 1, \ldots, R \\
j &= 1, \ldots, C \\
R &= \text{number of rows, or replicates in each factor} \\
C &= \text{number of columns, or factors} \\
\mu &= \text{overall average (mean)} \\
\tau_j &= \text{differential effect associated with } j\text{th level of } X \\
\epsilon_{ij} &= \text{noise or error associated with the } ij\text{th data value}
\end{aligned}$$

When discussing the sample statistics, $\bar{Y}_{.j}$ will refer to the mean of column j. For example,

$$\bar{Y}_{.1} = (Y_{11} + Y_{21} + \cdots + Y_{R1})/R$$

Meanwhile, $\bar{Y}_{..}$ will refer to the overall sample mean. Other notation worth mentioning are **TSS** for **total sum of squares**, **SSBc** for **sum of squares between columns** and **SSW** for **sum of squares within columns**. **SSW** are equivalent to **SSE**, the **error sum of squares**.

2.2 Reading Datasets into R

For our purpose, we shall use example 2.2 from ED. It investigates how battery lifetimes vary according to the devices they are used on.

The first step to carrying out data analyses with R is, obviously, to read in the data. Although R can handle data of several formats, it prefers a simple text file, with two columns - one for the dependent variable and another for the independent one. If this is available, the easiest way to read in data is with the `read.table()` function. Assuming the file name is *eg2.2*, this is the required syntax.

```
> battlife <- read.table("eg2.2")
```

It is natural for the first line in datasets to denote the names of the columns. For example, the first 5 lines in *eg2.2* are

hrs	device
1.8	dev1
5.0	dev1
1.0	dev1
4.2	dev2

To cater to this, use the following option

```
> battlife <- read.table("eg2.2", header = T)
```

Other times, we may leave comments at the beginning of the file to remind ourselves where the data came from, when it was collected and so on. You can tell R to ignore these comments, but you need to know how many lines these comments take up.

```
> battlife <- read.table("eg2.2", header = T, skip = 3)
```

The above would tell R to skip the first 3 lines of the file , read in the header, and then the data proper.

R has now stored the data in an object named `battlife`. Having read in the header, we can access and work on individual columns with `battlife$device` and `battlife$hrs`. The mean and standard deviation of the whole dataset can be obtained with the following commands.

```
> mean(battlife$hrs)

[1] 5.8

> sd(battlife$hrs)

[1] 2.244220
```

To obtain the mean lifetime of only the batteries used with device 8,

```
> mean(battlife$hrs[battlife$device == "dev8"])

[1] 7.4
```

2.3 The `aov()` function

We first have to create an `aov` object, which we name `mod1`, with the following command.

```
> mod1 <- aov(hrs ~ device, data = battlife)
```

The first argument tells R that `hrs` is the dependent variable and `device` is the independent one. The second argument is to tell R to look for these variables in the object `battlife`. To print out the analysis in a pretty format, use

```
> summary(mod1)

            Df Sum Sq Mean Sq F value  Pr(>F)
device       7 69.120   9.874  3.3816 0.02064 *
Residuals   16 46.720   2.920
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The entries in the table are easily understood. The entry in the rightmost column is the p-value. If instead we wish to test at a fixed significance level, say at the $\alpha = 0.075$ level, we can find the critical F value with

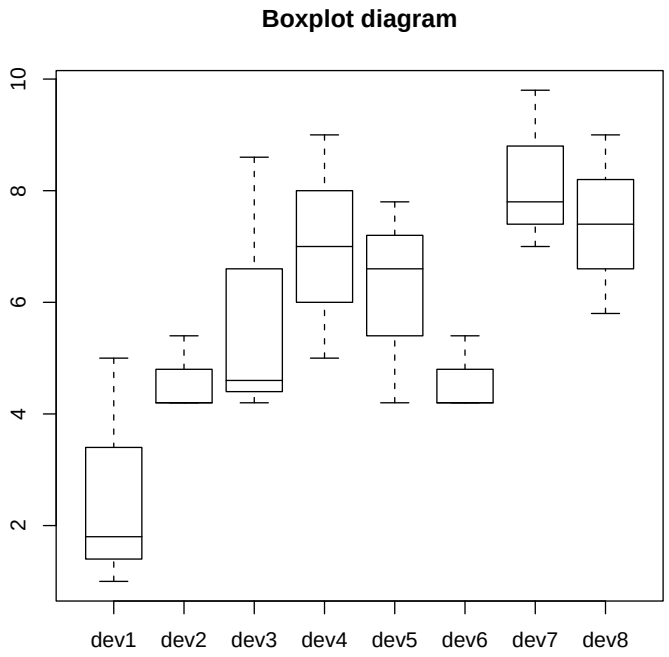

Figure 2.1: Boxplots for Battery Life Data

```
> qf(0.075, 7, 16, lower.tail = F)

[1] 2.344197
```

The above command informs R that we are interested in the 0.075-th quantile of the F distribution with 7 and 16 degrees of freedom. The last argument is crucial, because by default, R returns the quantile for the lower tail.

Another function useful in one-factor analyses is `boxplot()`. It allows for a visual comparison of the data obtained at the different levels.

```
> boxplot(hrs ~ device, data = battlife,
+ main = "Boxplot diagram")
```

The output from R is shown in Figure 2.1. The first two arguments to the command are the same as those used above with `aov()`, while the final argument provides the title of the plot. It can be altered according to your preference.

Chapter 3

Some Further Issues in One-Factor Designs and ANOVA

This chapter is really an extension to the previous one, because whenever we draw from inferences from an ANOVA, we have to check if any assumptions have been violated. The next section covers these tests. If the assumptions are severely violated, it might be appropriate to carry out a non-parametric test. The Kruskal-Wallis test, which is available in R is covered here. Finally, we include instructions on how to construct confidence intervals. The dataset used is the battery life one, hence we assume the model has already been built, as per Section 2.3.

3.1 Testing the Validity of the Assumptions

The three main assumptions made when conducting ANOVA tests are

1. The residuals ϵ_{ij} are independent and uncorrelated random variables.
2. The residuals ϵ_{ij} are normally distributed.
3. The residuals ϵ_{ij} have a mean 0 and a constant variance.

The first assumption is sensitive, while the latter two are more robust. Assumptions 1 and 3 are usually checked via graphical plots, while assumption 2 is usually checked through a normal-normal quantile plot.

Before proceeding with the plots, it has to be pointed out that the `aov` object created contains more than just a method to print out the ANOVA table. All the data that it holds can be listed with the following command.

```
> names(mod1)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "contrasts"     "xlevels"       "call"          "terms"
[13] "model"
```

The individual elements can then be accessed with the `$` operator. For example, to list all the residuals, use

```
> mod1$residuals
```

```
            1             2             3             4             5
-8.000000e-01  2.400000e+00 -1.600000e+00 -4.000000e-01  8.000000e-01
            6             7             8             9            10
-4.000000e-01  2.800000e+00 -1.200000e+00 -1.600000e+00  4.227033e-17
           11            12            13            14            15
-2.000000e+00  2.000000e+00 -2.000000e+00  1.600000e+00  4.000000e-01
           16            17            18            19            20
-4.000000e-01 -4.000000e-01  8.000000e-01 -4.000000e-01 -1.200000e+00
           21            22            23            24
 1.600000e+00  1.600000e+00  6.528930e-16 -1.600000e+00
```

Now, back to the plots. There will be 4 of them, and in order to save space they will be carried out on a single graphical device. This will be acheived with the help of the `split.screen()` command in R. The following commands are to open a graphics device, split it into 4, then select the first screen to be drawn on.

```
> x11()
> split.screen(c(2, 2))

[1] 1 2 3 4

> screen(1)
```

The first 3 plots are all for checking the assumptions 1 and 3. They consist of plotting the residuals against the fitted values, the factor levels and in time order. The last plot is quite meaningless in the battery life example, but it is included for completeness' sake. The fourth plot is the normality check.

```
> plot(mod1$fitted.values, mod1$residuals,
+ main = "Residuals vs. Fitted", pch = 20)
> abline(h = 0, lty = 2)
> screen(2)
> plot(mod1$model$device, mod1$residuals,
```

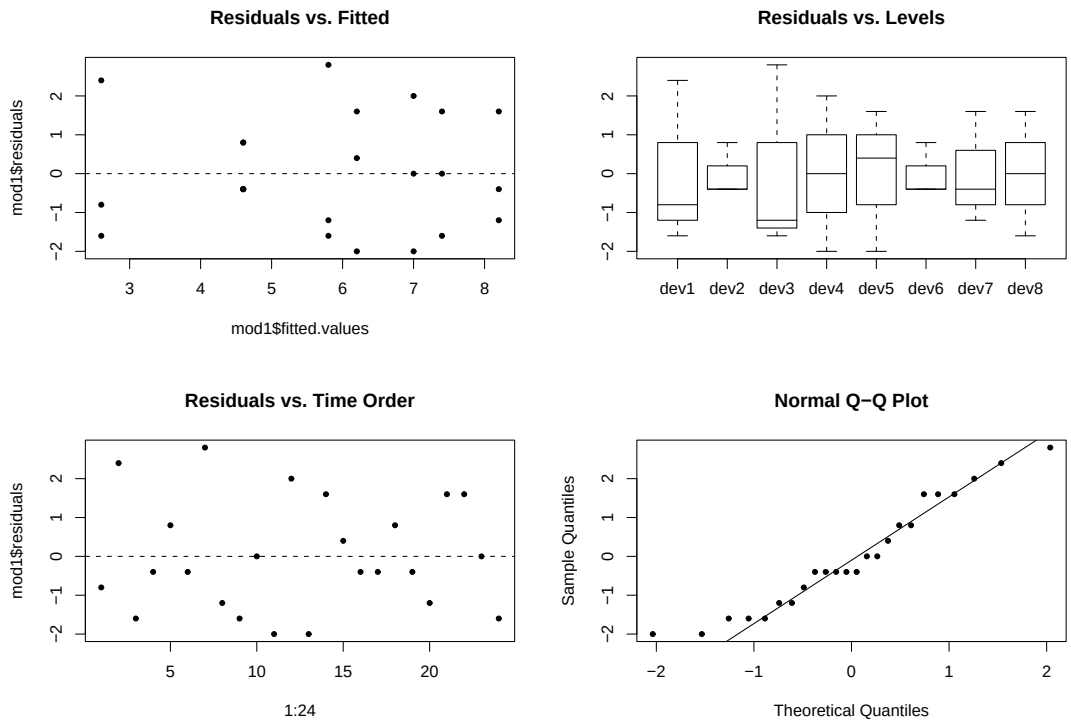

Figure 3.1: Residual Plots

```
+ main = "Residuals vs. Levels")
> screen(3)
> plot(1:24, mod1$residuals, main = "Residuals vs. Time Order",
+     pch = 20)
> abline(h = 0, lty = 2)
> screen(4)
> qqnorm(mod1$residuals, pch = 20)
> qqline(mod1$residuals)
```

3.2 Kruskal-Wallis Test

As mentioned earlier, this is a non-parametric test for use when the above assumptions have been significantly violated. It is quite easy to run; just two lines of code are needed.

```
> mod2 <- kruskal.test(battlife$hrs, battlife$device)
> mod2

        Kruskal-Wallis rank sum test
```

```
data:  battlife$hrs and battlife$device
Kruskal-Wallis chi-squared = 13.0096, df = 7, p-value = 0.07188
```

3.3 Confidence Intervals

3.3.1 For the Mean at a Specific Level

Confidence intervals can be constructed for each of the levels at which the experiment was carried out. The formula is given by

$$\bar{X} \pm t_{1-\alpha/2} \left(s/\sqrt{n} \right)$$

Let us take the terms, one at a time from left to right. All of them will be derived from the `mod1` model that was built earlier. Returning to the example on battery life, assume that we wish to compute the confidence interval for batteries used on device 6. Obtaining $\bar{X}$ is quite straightforward.

```
> xbar <- mean(mod1$model$hrs[mod1$model$device == "dev6"])
```

The critical t value can be obtained just as the quantile for the F distribution was obtained in Section 2.3. Remember that the degrees of freedom follows the SSE degrees of freedom. To avoid making any careless mistakes, use the following command to obtain the required quantile to construct a 95% confidence interval.

```
> tcrit <- qt(0.025, mod1$df.residual, lower.tail = F)
```

For s, we need to find the square root of MSW. Since MSW is the sum of the square of the residuals divided by the degrees of freedom, there is a simple formula we can apply.

```
> s <- sqrt(sum((mod1$residuals)^2)/mod1$df.residual)
```

Finally, n comes from the number of data points at the level of device 6. Here is a general way to find this number.

```
> n <- sum(mod1$model$device == "dev6")
```

We can put them all together to obtain the lower and upper confidence intervals simultaneously.

```
> xbar - tcrit * (s/sqrt(n)); xbar + tcrit * (s/sqrt(n))

[1] 2.508551
[1] 6.691449
```

3.3.2 For the Difference Between Two Means

The formula to derive the confidence intervals for the difference between two means is

$$(\bar{X}_1 - \bar{X}_2) \pm t_{1-\alpha/2} \left(s\sqrt{1/n_1 + 1/n_2} \right)$$

Say we wish to find the interval for the difference between levels at devices 1 and 3. The inputs s and $t_{1-\alpha/2}$ have already been computed in Section 3.3.1. The remaining values can be obtained with

```
> n1 <- sum(mod1$model$device == "dev1")
> n2 <- sum(mod1$model$device == "dev3")
> x1bar <- mean(mod1$model$hrs[mod1$model$device == "dev1"])
> x2bar <- mean(mod1$model$hrs[mod1$model$device == "dev3"])
```

Substituting into the formula will yield the required interval.

```
> (x1bar - x2bar) - tcrit * (s * sqrt(1/n1 + 1/n2)) ; (x1bar
+ -x2bar) + tcrit * (s * sqrt(1/n1 + 1/n2))

[1] -6.157755
[1] -0.2422446
```

Chapter 4

Multiple-Comparison Testing

ED covers two kinds of multiple-comparison tests in this chapter - pairwise and post hoc. A priori tests are covered in the next chapter on orthogonality. ED covers 4 pairwise tests. However, R has only a function for Tukey's Honestly Significant Difference (HSD) test. Although the rest of the tests are not included in R, they can still be carried out in a few steps. In this document, we cover how to compute Fisher's Least Significant Difference (LSD) and how to use the `TukeyHSD()` function.

4.1 Fisher's Least Significant Difference Test

The LSD test essentially involves performing a series of pairwise t tests. It is important to note that with LSD, we have to specify the **individual** error rate, not the experimentwise one. The example used in this chapter is the broker study. The data can be found in the file *eg4.3*. Here are the steps to create `mod2`.

```
> brok <- read.table("eg4.3", header = T)
> mod2 <- aov(index ~ broker, data = brok)
```

Now we can proceed to compute the LSD. In fact, we had actually computed the LSD when deriving the confidence intervals for the difference between two means in Section 3.3.2. Assume here that we wish to find the LSD between brokers 2 and 5.

```
> n1 <- sum(mod2$model$broker == "brok2")
> n2 <- sum(mod2$model$broker == "brok5")
> s <- sqrt(sum((mod2$residuals)^2)/mod2$df.residual)
> tcrit <- qt(0.025, mod2$df.residual, lower.tail = F)
> LSD <- tcrit * s * sqrt(1/n1 + 1/n2)
> LSD
```

```
[1] 5.474913
```

The last two lines compute the LSD and display it.

4.2 Tukey's Honestly Significant Difference Test

The HSD test focuses on the experimentwise error rate, **a**, and assumes that the number of replicates at each level are equal. To use the function, an `aov` object is required as the first argument.

```
> mod2.tukey <- TukeyHSD(mod2, ordered = T)
> mod2.tukey

  Tukey multiple comparisons of means
    95% family-wise confidence level
    factor levels have been ordered

Fit: aov(formula = index ~ broker, data = brok)

$broker
             diff        lwr       upr
brok1-brok3     1 -6.8071443  8.807144
brok2-brok3     7 -0.8071443 14.807144
brok4-brok3     9  1.1928557 16.807144
brok5-brok3    12  4.1928557 19.807144
brok2-brok1     6 -1.8071443 13.807144
brok4-brok1     8  0.1928557 15.807144
brok5-brok1    11  3.1928557 18.807144
brok4-brok2     2 -5.8071443  9.807144
brok5-brok2     5 -2.8071443 12.807144
brok5-brok4     3 -4.8071443 10.807144
```

Requesting that R order the difference makes it easy to read. Check to see if 0 falls within the confidence intervals to determine if the difference between the means is significant. From the first line alone, it can be inferred that brokers 3 and 1 are in the same group. If you wish to compute the HSD at the **a**= 0.05 level explicitly, use this.

```
> qcrit <- qtukey(0.05, length(mod2$xlevels[[1]]),
+ mod2$df.residual, lower.tail = F)
> hsd <- qcrit * s/sqrt(6)
> hsd

[1] 7.807144
```

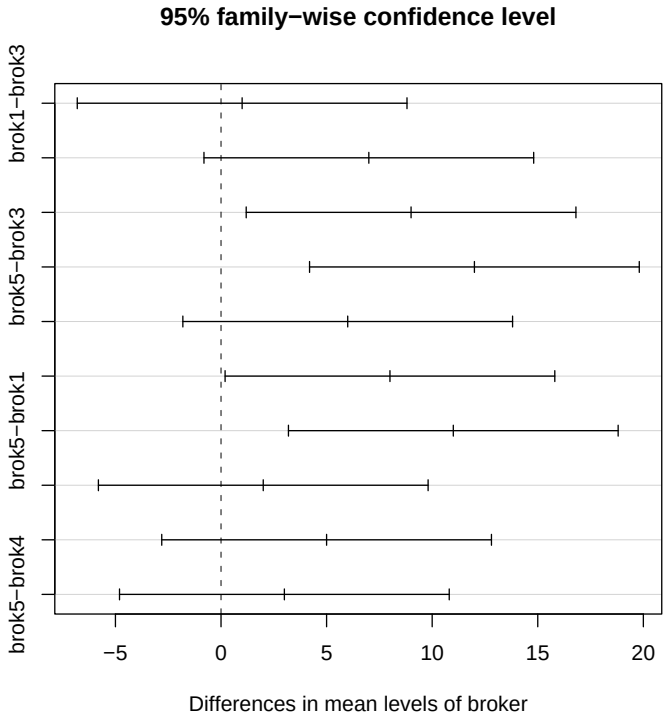

Figure 4.1: Tukey's HSD plot

Another neat trick R can do is to plot the confidence intervals. Note that the order of the intervals follows the same listing as the text printout.

```
> plot(mod2.HSD)
```

Chapter 5

Orthogonality, Orthogonal Decomposition, and Their Role in Modern Experimental Design

In the last chapter, ED covered two multiple-comparison techniques. Here a more versatile test is covered - orthogonal contrasts. The next section will cover setting up the contrasts matrix, after which we will use two examples from the text to illustrate their use in R. Fortunately, we have left the battery life example behind for good.

5.1 Setting up the Contrasts Matrix

ED states that there are typically $C - 1$ orthogonal contrasts when there are C levels to the factor. These can be arranged in the rows of a matrix, given that they abide by the following restrictions.

1. The dot product of every pair of different rows is 0.
2. The dot product of every row with itself is 1.
3. The sum of the entries in each row is 1.

An example of an orthonormal matrix, when there are 4 levels of the factor, is given below.

$$\begin{pmatrix} 1/2 & 1/2 & -1/2 & -1/2 \\ 1/2 & -1/2 & 1/2 & -1/2 \\ 1/2 & -1/2 & -1/2 & 1/2 \end{pmatrix}$$

There are countless ways to create a `matrix` object in R. As it is not our main focus, we simply cover one of them here.

```
> con <- matrix(c(0.5, 0.5, -0.5, -0.5, 0.5, -0.5, 0.5, -0.5,
+  0.5, -0.5, -0.5, 0.5), nrow = 4, ncol = 3, byrow = F)
> con

     [,1] [,2] [,3]
[1,]  0.5  0.5  0.5
[2,]  0.5 -0.5 -0.5
[3,] -0.5  0.5 -0.5
[4,] -0.5 -0.5  0.5
```

Notice that the contrasts are stored column-wise instead of row-wise. We are not deliberately trying to confuse the reader. It is merely that R prefers the contrasts this way. One could always create the constrasts matrix row-wise and then apply `t()` to transpose it.

5.2 Portfolio Rating and Drug Comparison Examples

We focus first on the portfolio rating example. The dataset is available in file *eg5.3*. After reading in the dataset, the next thing to do is not to build the model straightaway, but to set the contrast according to the matrix created earlier. The intuitive command is `contrasts()`.

```
> pf <- read.table("eg5.3", header = T)
> contrasts(pf$portfolio) <- con
```

Now when we build the model, R will note that we have specified a certain set of contrasts. When we invoke the `summary.aov()` method, we can recall the contrasts according to their columns - 1, 2 and 3. It is of great consequence that the exact name of the factor column be used when invoking the `summary.aov()` method. In this case it is "portfolio".

```
> mod3 <- aov(rating ~ portfolio, data = pf)
> summary.aov(mod3, split = list(portfolio = list(
+ aggressiveness = 1, balance = 2, environ = 3)))

                           Df  Sum Sq Mean Sq F value    Pr(>F)
portfolio                   3 184.000  61.333 20.4444 5.220e-05 ***
  portfolio: aggressiveness  1 144.000 144.000 48.0000 1.587e-05 ***
  portfolio: balance         1  36.000  36.000 12.0000  0.004682 **
  portfolio: environ         1   4.000   4.000  1.3333  0.270690
Residuals                   12  36.000   3.000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now let us move on to the drug comparison example, whose data can be obtained from the *eg5.4* file.

```
> dg <- read.table("eg5.4",header=T)
> levels(dg$drug)

[1] "asp1"    "asp2"    "nonA"    "placebo"
```

Notice that the levels have been ordered alphabetically, meaning that the ordering is different from the ED book. Taking this into account, we have to set up the contrasts matrix carefully and correctly.

```
> con <- matrix(c(0.2887, 0.2887, 0.2887, -0.866, -0.7071,
+ 0.7071, 0, 0, -0.4082, -0.4082, 0.8165, 0), 4, 3)
> contrasts(dg$drug) <- con
> mod4 <- aov(improvement ~ drug, data = dg)
> mod4$contrasts

$drug
           [,1]    [,2]    [,3]
asp1     0.2887 -0.7071 -0.4082
asp2     0.2887  0.7071 -0.4082
nonA     0.2887  0.0000  0.8165
placebo -0.8660  0.0000  0.0000
```

The last command was to check that the contrasts have been set up correctly and that the three questions of interest are being answered (via the columns of the matrix): P versus P', A1 versus A2 and A versus Non-A. Now we can ask R to split the SSB for us.

```
> summary.aov(mod4, split = list(drug = list("P vs. P'" = 1,
+ "A1 vs. A2" = 2, "A vs. Non-A" = 3)))

                   Df  Sum Sq Mean Sq F value    Pr(>F)
drug                3 112.000  37.333  7.4667 0.0008018 ***
  drug: P vs. P'    1  42.667  42.667  8.5333 0.0068202 **
  drug: A1 vs. A2   1   4.000   4.000  0.8000 0.3787177
  drug: A vs. Non-A 1  65.333  65.333 13.0667 0.0011681 **
Residuals          28 140.000   5.000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Chapter 6

Two Factor Cross-Classification Designs

In the previous chapters, we dealt exclusively with experiments with only one factor. This is a good starting point, but highly unrealistic. This chapter introduces the reader to experiments with two factors. ED returns to the battery life example again to illustrate two-way ANOVA tests. Just as in the one-factor case, the assumptions are then tested and a non-parametric test is demonstrated as an alternative.

6.1 Two Factor ANOVA

It is instructive to state the statistical model before going further.

$$Y_{ijk} = \mu + \rho_i + \tau_j + + I_{ij} + \epsilon_{ijk}$$

where

$$\begin{aligned}
i &= 1, \ldots, R \\
j &= 1, \ldots, C \\
k &= 1, \ldots, n \\
R &= \text{number of rows} \\
C &= \text{number of columns} \\
n &= \text{number of replicates at each treatment level} \\
\mu &= \text{overall average (mean)} \\
\rho_i &= \text{differential effect associated with the } i\text{th row factor} \\
\tau_j &= \text{differential effect associated with the } j\text{th column factor} \\
I_{ij} &= \text{interaction effect associated with the } i\text{th row and } j\text{th column factor} \\
\epsilon_{ijk} &= \text{noise or error associated with the } ijk\text{th data value}
\end{aligned}$$

We can dive straight into the battery life data, which now has a second factor, `brand`. The dataset is in the file *eg6.2*.

```
> battlife <- read.table("eg6.2", header = T)
> mod5 <- aov(hrs ~ device * brand, data = battlife)
> summary(mod5)

             Df   Sum Sq  Mean Sq F value  Pr(>F)
device        2 0.280000 0.140000  5.6000 0.01915 *
brand         3 0.210000 0.070000  2.8000 0.08533 .
device:brand  6 0.110000 0.018333  0.7333 0.63251
Residuals    12 0.300000 0.025000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

R is told to include the "`device*brand`" interaction term, and hence will include the main effects `device` and `brand` by default. ED points out that the interaction effects are not significant. For further evidence, we can draw an interaction plot.

```
> interaction.plot(battlife$device, battlife$brand, battlife$hrs,
+ type = "l", main = "Battery Life Interaction Plot", xlab =
+ "Device", ylab = "Hours", trace.label = "brand", col = 1:4)
```

The first argument to the function denotes the levels that will be plotted on the x-axis. The second tells R how many lines to draw on the plot. In this case, there are 4 brands, resulting in 4 lines. The third argument is the response, which will be plotted on the y-axis. By default, R plots the mean of each treatment combination, that is, $\bar{Y}_{ij\cdot}$. In the above, we have also requested R to plot lines (by joining the dots), to label the axes as we have specified and to use 4 different colours.

As the ANOVA output and the interaction plot in Figure 6.1 both seem to indicate that the interaction effect is insignificant, ED recommends pooling the SSE and SSI. This can be easily achieved with

```
> mod5a <- aov(hrs ~ device + brand, data = battlife)
> summary(mod5a)

            Df  Sum Sq Mean Sq F value   Pr(>F)
device       2 0.28000 0.14000  6.1463 0.009234 **
brand        3 0.21000 0.07000  3.0732 0.054089 .
Residuals   18 0.41000 0.02278
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

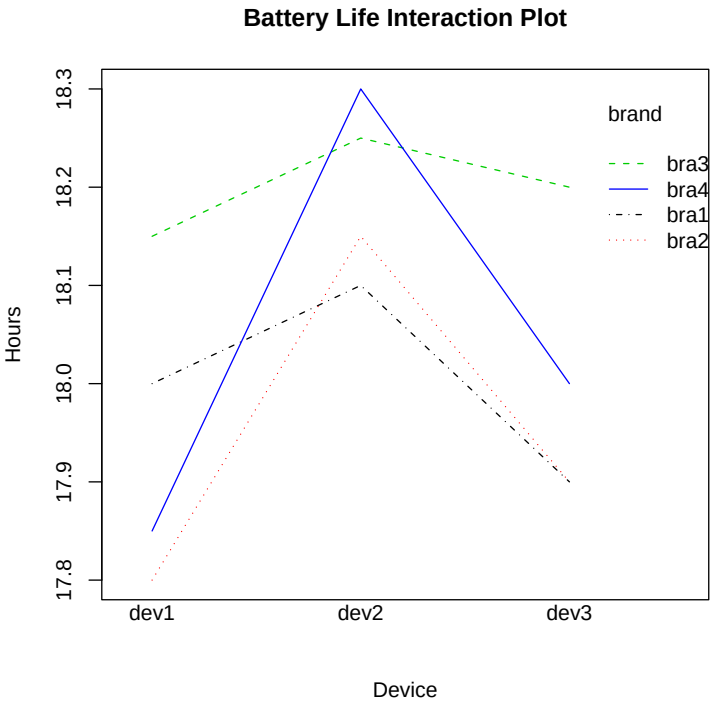

Figure 6.1: Interaction Plot

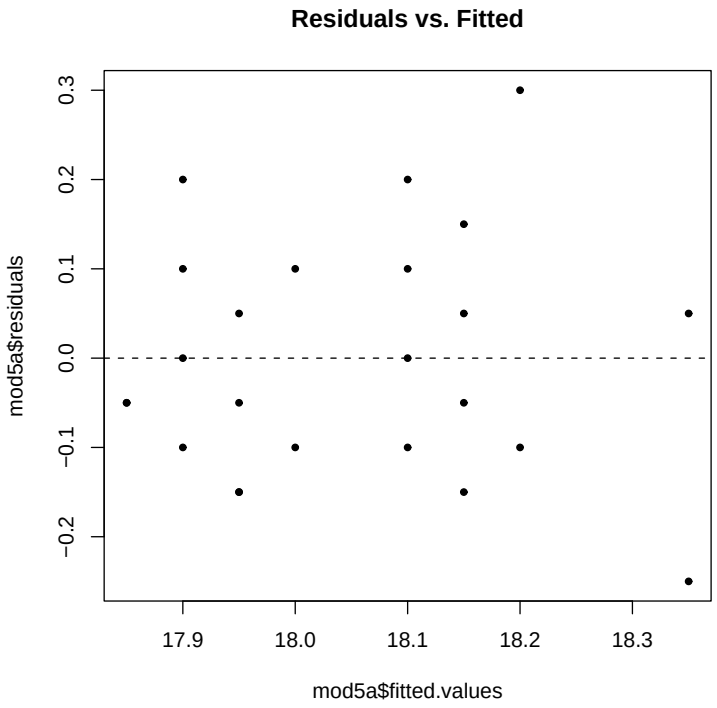

Figure 6.2: Residual Plot

It always good to check the residuals in our model. Here we only dislay two plots - in Figures 6.2 and 6.3. The remaining can be drawn the same way as in section 3.1.

```
> plot(mod5a$fitted.values, mod5a$residuals,
+ main = "Residuals vs. Fitted", pch = 20)
> abline(h = 0, lty = 2)
> qqnorm(mod5a$residuals, pch = 20)
> qqline(mod5a$residuals)
```

6.2 Two Factors with No Replication and No Interaction

When there are no replicates, ED tells us that we have to assume there is no interaction and use the SSI as the SSE. To tell R to do this, simply create a model with only main effects. The rest will lumped with the SSE automatically. The data for this example can be found in the file *eg6.8*.

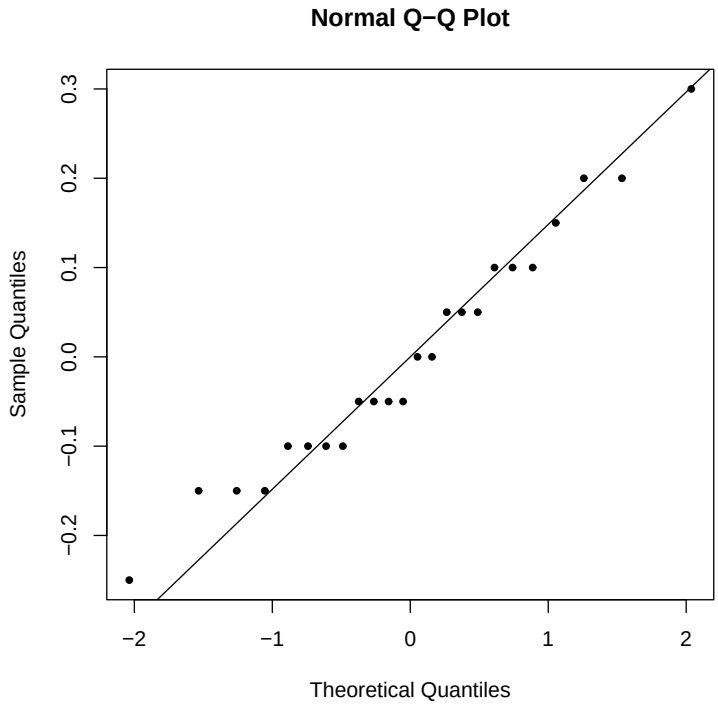

Figure 6.3: Normal Quantile Plot

```
> unrep <- read.table("eg6.8", header = T)
> mod6 <- aov(resp ~ A + B, data = unrep)
> summary(mod6)

            Df Sum Sq Mean Sq F value    Pr(>F)
A            2 32.000  16.000      72   6.4e-05 ***
B            3 28.667   9.556      43 0.0001888 ***
Residuals    6  1.333   0.222
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Naturally, the assumptions should be checked via the residuals, and if they are severly violated, the Friedman nonparametric test (see below) should be carried out.

6.3 Friedman Nonparametric Test

Remember that this test is for two factor ANOVA without replication. The required function in R is `friedman.test()`. The dataset for this angioplasty example is in *eg6.11*. The function can be called in three different ways. All give the same output, so it depends on what format your dataset is in. In our case, since our responses are all in a vector, we can use 2 of the methods directly.

The first method involves specifying the response vector, group vector (which we wish to test) and the block factor in the first 3 arguments. In this example, our groups are angioplasty unit type and the blocks are the balloon dilation catheter type. The response is the bursting pressure.

```
> angi <- read.table("eg6.11", header = T)
> friedman.test(angi$press, angi$angio, angi$balloon)

        Friedman rank sum test

data:  angi$press, angi$angio and angi$balloon
Friedman chi-squared = 20.7241, df = 3, p-value = 0.0001201
```

The second method requires the formula and the dataset to specified.

```
> friedman.test(press ~ angio | balloon, data = angi)

        Friedman rank sum test

data:  press and angio and balloon
Friedman chi-squared = 20.7241, df = 3, p-value = 0.0001201
```

The last method requires manipulation of our dataset to form a matrix of the response. Data will be in the columns according to the factor we desire to test. The rows will contain the block factor.

```
> x1 <- angi[angi$angio == "typeA", 1]
> x2 <- angi[angi$angio == "typeB", 1]
> x3 <- angi[angi$angio == "typeC", 1]
> x4 <- angi[angi$angio == "typeD", 1]
> angi_data <- cbind(x1, x2, x3, x4)
> angi_data

      x1 x2 x3 x4
 [1,] 24 26 25 22
 [2,] 27 27 26 24
 [3,] 19 22 20 16
 [4,] 24 27 25 23
 [5,] 22 25 22 21
 [6,] 26 27 24 24
 [7,] 27 26 22 23
 [8,] 25 27 24 21
 [9,] 22 23 20 19
```

Taking a look at the dataset, we see that it is identical to the table diplayed in the textbook. Now we can demonstrate the final way of calling the `friedman.test()` function.

```
> friedman.test(angi_data)

        Friedman rank sum test

data:  angi_data
Friedman chi-squared = 20.7241, df = 3, p-value = 0.0001201
```

Chapter 7

Nested, or Hierarchical Designs

At first look, nested designs look the same as two factor designs. It is crucial to understand the difference though. With two factor designs, each level of factor B (the row) stays the same across all levels of factor A (the column). Whereas with nested designs, the level of factor B differs across factor A. Using the example from ED, if it were a two factor experiment, the same professor would teach all statistical packages. However, since it is a nested design, different professors teach the different packages.

7.1 The Model and its Analysis

It will also be clear from the model that there is no interaction term in a nested model. The reason being that the levels of the minor factor have no link to the other levels of their major factor.

$$Y_{ijk} = \mu + \rho_i + \tau_{(i)j} + \epsilon_{(ij)k}$$

where

$$\begin{aligned} i &= 1, 2, 3, \ldots, M \\ j &= 1, 2, 3, \ldots, m \\ k &= 1, 2, 3, \ldots, n \\ M &= \text{number of levels of the major factor} \\ m &= \text{number of levels of the minor factor} \\ k &= \text{number of replicates per } (i, j) \text{ combination} \end{aligned}$$

Building the model is really quite straightforward in R. It is a matter of specifying the right formula for R to interpret. The dataset for this example is in *eg7.3*.

```
> stud <- read.table("eg7.3", header = T)
> mod7 <- aov(score ~ statspkg + statspkg:professor, data = stud)
> summary(mod7)

                   Df Sum Sq Mean Sq F value   Pr(>F)
statspkg            3 306.30  102.10  6.9080 0.005902 **
statspkg:professor  8 162.56   20.32  1.3748 0.298572
Residuals          12 177.36   14.78
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The never-before-seen term is the one with the colon. It informs R that the levels of `professor` are nested within `statspkg`.

Chapter 8

Designs with Three or More Factors: Latin-Square and Related Designs

Latin square designs are used when we have 3 factors, each with the same number of levels, say m, but do not wish to carry out the full experiment with m^3 treatment combinations. Instead, we can get by with just m^2 combinations. When we have more than 3 factors, the related Graeco-Latin square designs are used. Since we are not carrying out the full experiment, we have to make the assumption that there is no, or only negligible, interaction between the factors. In upcoming chapters, ED will cover two level factorial designs, where this assumption need not be made and where the interaction effects of interest can still be estimated.

First though, what is a Latin square? It is a matrix of size m by m, with the entries in each row and column being 1 to m. Moreover, each number appears exactly once in each row, and exactly once in each column. A 4x4 example is given below.

2	4	3	1
3	1	2	4
4	3	1	2
1	2	4	3

This is in fact the Latin square on which the valet-parking use example in ED is based. The dataset is available in *eg8.2*. Once you have yourself a Latin square, use the values in the matrix to index your "inside" factor, and add the row and column factors to arrive at your experimental set-up.

	B_1	B_2	B_3	B_4
A_1	C_2	C_4	C_3	C_1
A_2	C_3	C_1	C_2	C_4
A_3	C_4	C_3	C_1	C_2
A_4	C_1	C_2	C_4	C_3

8.1 Latin Square Analysis with R

As mentioned earlier, the model will not include an interaction term.

$$Y_{ijk} = \mu + \rho_i + \tau_j + \gamma_k + \epsilon_{ijk}$$

where i, j and k all run from 1 to m. Analysing the data is quite similar to the designs in previous chapters.

```
> valuse <- read.table("eg8.2", header = T)
> mod8 <- aov(pats ~ cost + spaces + valets, data = valuse)
> summary(mod8)

            Df Sum Sq Mean Sq F value    Pr(>F)
cost         3  370.5   123.5  2.9173  0.122507
spaces       3 9025.0  3008.3 71.0630 4.441e-05 ***
valets       3 1389.5   463.2 10.9409  0.007591 **
Residuals    6  254.0    42.3
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

8.2 Other useful R Functions for ANOVA

Before moving on to Part II of ED, here we cover a couple of functions that might be useful but do not fall into any specific chapter of ED. Firstly, the function `model.tables()` can be used to list the means at the different treatment combinations and the effects of the different factor levels. The first argument to the function has to be the result of a call to the `aov()` function, and the second specifies whether you desire the means or the estimated effects.

```
> model.tables(mod8, "means")

Tables of means
Grand mean

51.75
```

```
 cost
cost
 cos1  cos2  cos3  cos4
49.50 50.75 47.00 59.75

 spaces
spaces
spa1 spa2 spa3 spa4
87.5 59.0 31.0 29.5

 valets
valets
 val1  val2  val3  val4
40.50 48.50 51.75 66.25
```

```
> model.tables(mod8, "effects")
```

```
Tables of effects

 cost
cost
 cos1  cos2  cos3  cos4
-2.25 -1.00 -4.75  8.00

 spaces
spaces
  spa1   spa2   spa3   spa4
 35.75   7.25 -20.75 -22.25

 valets
valets
  val1   val2   val3   val4
-11.25  -3.25   0.00  14.50
```

The output from the second call can be interpreted according to the model in 8.1. Thus, $\hat{\rho}_1 = -2.25$, $\hat{\rho}_2 = -1.00$ and so on. The estimate for μ is given in the first line in the first function call, 51.75.

The second function of note, `replications()` is a quick way of checking how many replicates there are at each level in the model. The input to the function is the same as that to the `aov` call. The proposed model has to be fully specified. The function returns a list only if the model is unbalanced. Hence using `!is.list(replications(pats  cost + spaces + valets,data=valuse))` will allow you to test if the model is balanced.

```
> replications(pats ~ cost + spaces + valets, data = valuse)

  cost spaces valets
     4      4      4

> !is.list(replications(pats ~ cost + spaces +
+ valets, data = valuse))

[1] TRUE
```

With this, we come to an end to Part I of ED.

Part II

Primary Focus on the Number of Levels of a Factor

Chapter 9

Two-Level Factorial Designs

This chapter in the book is concerned primarily with demonstrating how to compute the effects, by hand, in a two level factorial experiment. However, tasks such as noting the Yates order, assigning plus and minus signs should be transparent to the user of a good software, such as R. Hence this chapter is rather short compared to the textbook. We take the opportunity to introduce another useful function in R, `plot.design()`.

9.1 Analysing Two-Level Factorial Designs

The dataset used is available in *eg9.5*. The levels of high and low are used in order to retain the terminology used in ED.

```
> rate <- read.table("eg9.5", header = T)
> mod9 <- aov(resp ~ postage * price * env_size, data = rate)
> summary(mod9)

                       Df    Sum Sq   Mean Sq
postage                 1 0.0003125 0.0003125
price                   1 0.0047045 0.0047045
env_size                1 0.0000720 0.0000720
postage:price           1 0.0000720 0.0000720
postage:env_size        1 0.0000045 0.0000045
price:env_size          1 0.0000405 0.0000405
postage:price:env_size  1 0.0000500 0.0000500
```

Since there is only one replicate at each treatment combination, no F-values are computed. The magnitude of the estimated effects can be viewed with

```
> model.tables(mod9, "effects")
```

```
Tables of effects

 postage
       high      low
    0.00625 -0.00625
rep 4.00000  4.00000

 price
        high     low
    -0.02425 0.02425
rep  4.00000 4.00000

 env_size
     high    low
    0.003 -0.003
rep 4.000  4.000

 postage:price
       price
postage high   low
   high -0.003  0.003
   rep   2.000  2.000
   low   0.003 -0.003
   rep   2.000  2.000

 postage:env_size
       env_size
postage high   low
   high  7e-04 -7e-04
   rep   2e+00  2e+00
   low  -7e-04  7e-04
   rep   2e+00  2e+00

 price:env_size
      env_size
price  high    low
  high  0.0022 -0.0022
  rep   2.0000  2.0000
  low  -0.0022  0.0022
  rep   2.0000  2.0000

 postage:price:env_size
, , env_size = high
```

```
       price
postage high    low
   high -0.0025  0.0025
   rep   1.0000  1.0000
   low   0.0025 -0.0025
   rep   1.0000  1.0000

, , env_size = low

       price
postage high    low
   high  0.0025 -0.0025
   rep   1.0000  1.0000
   low  -0.0025  0.0025
   rep   1.0000  1.0000
```

The values are half of what are computed in ED because R provides the change in response per unit change in the factor level, whereas the book provides the change in response per two-unit change in the factor level. To obtain a plot of the mean response at the various factor levels, use `plot.design`.

```
> plot.design(resp ~ postage * price * env_size,
+ data = rate, main = "Main Effects Plot")
```

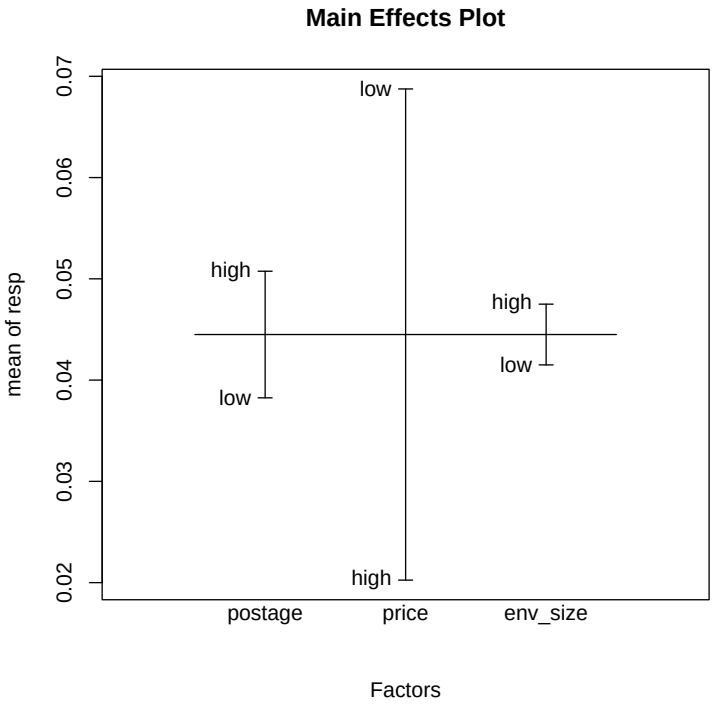

Figure 9.1: Mean Response By Factor

Chapter 10

Confounding/Blocking in 2^k Designs

In our opinion, this chapter is where we begin to see the true beauty of experimental design. The ability to choose which effect(s) we wish to confound is very useful when we cannot run all the treatments under the same conditions. Computing of F-values are shown in Section 10.3.

The first two sections of this chapter are on choosing which effects to confound and finding out which effects are confounded by our set-up. To do this, it is necessary to install an add-on package for R. This is described next. Remember that installation needs to be carried out as the root user under *NIX environments and as the Administrator under WinXP.

10.1 Simple Confounding

The required package is called `conf.design()`. The first of the 3 steps below is to download and install the packages. The second is to attach the new library for this session. Lastly, we can read the online documentation with the "?" operator.

```
> install.packages("conf.design")
> library(conf.design)
> ?conf.design
```

Simple confounding occurs when only one effect is confounded, implying that the treatments have been split into 2 blocks. The function `conf.design()` will describe how the treatments should be split according to the specified confounded effect. Suppose we have a 2^3 factorial experiment, which we need to run in 2 blocks. Assume also that we do not mind being unable to estimate the effect of factor C. In other words, the effect of C will be confounded.

```
> conf.design(c(0, 0, 1), p = 2, block.name = "blk",
+ treatment.names = c("A", "B", "C"))

  blk A B C
1   0 0 0 0
2   0 1 0 0
3   0 0 1 0
4   0 1 1 0
5   1 0 0 1
6   1 1 0 1
7   1 0 1 1
8   1 1 1 1
```

R tells us that treatments $1, a, b$ and ab should be run in block 0 and the rest in block 1. The first input to the function is a vector specifying the effect we wish to confound, in this case C. R will deduce the number of factors from here. In Section 10.2 below, when we carry out multiple confounding, this first entry will be a matrix. The second argument is the number of levels of each factor. Third is a name for the blocks. Last of all comes the names of the factors.

If we wish to carry out partial confounding, we merely have to specify the different effects we wish to confound and run the function several times.

10.2 Multiple Confounding

Multiple confounding is carried out when the number of blocks is more than 2. In such situations it is crucial to know how many effects must be specified and how many will be confounded in total. Imagine now we wish to run a 2^5 factorial experiment in 4 blocks of 8 treatment combinations each. In total 3 effects will be confounded, and we as the designers only have the leeway to specify 2 of these three. Suppose we pick the effects AB and BCD to throw away.

```
> G <- rbind(c(1, 1, 0, 0, 0), c(0, 1, 1, 1, 0))
> conf.design(G, p = 2, block.name = "blk",
+ treatment.names = c("A", "B", "C", "D", "E"))

   blk A B C D E
1   00 0 0 0 0 0
2   00 1 1 1 0 0
3   00 1 1 0 1 0
4   00 0 0 1 1 0
5   00 0 0 0 0 1
```

```
6   00 1 1 1 0 1
7   00 1 1 0 1 1
8   00 0 0 1 1 1
9   01 1 1 0 0 0
10  01 0 0 1 0 0
11  01 0 0 0 1 0
12  01 1 1 1 1 0
13  01 1 1 0 0 1
14  01 0 0 1 0 1
15  01 0 0 0 1 1
16  01 1 1 1 1 1
17  10 1 0 0 0 0
18  10 0 1 1 0 0
19  10 0 1 0 1 0
20  10 1 0 1 1 0
21  10 1 0 0 0 1
22  10 0 1 1 0 1
23  10 0 1 0 1 1
24  10 1 0 1 1 1
25  11 0 1 0 0 0
26  11 1 0 1 0 0
27  11 1 0 0 1 0
28  11 0 1 1 1 0
29  11 0 1 0 0 1
30  11 1 0 1 0 1
31  11 1 0 0 1 1
32  11 0 1 1 1 1
```

The first line is to specify the two effects in the form of a matrix. The next command will output the desired blocks. When we only need to list all the confounded effects, we can call on `conf.set()` and pass the effects we have chosen as the argument. In the above example, we pass AB and BCD as a matrix and the function will list all the 3 confounded effects.

```
> conf.set(G, p = 2)

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    0    0    0
[2,]    0    1    1    1    0
[3,]    1    0    1    1    0
```

10.3 Analysing Block Designs with R

The textbook ED does not demonstrate any example on the above, so we have to make do with a purely pedantic dataset. It can be found in the file

eg10. It is from a 2^3 full factorial experiment with 2 replicates. It is run in two blocks with the confounded effect being ABC. To run the model just remember that when we split the treatments into blocks, we have assumed that there is no interaction between the block factor and the other factors of interest. Thus when specifying the formula, the block is purely additive. Having replicates will allow us to estimate the error. Without it, all degrees of freedom will be used up on the effects.

```
> library(conf.design)
> dat <- read.table("eg10", header = T)
> mod10 <- aov(resp ~ block + A * B * C, data = dat)
> summary(mod10)

            Df    Sum Sq   Mean Sq   F value    Pr(>F)
block        1 1.808e-37 1.808e-37 1.647e-33    1.0000
A            1 5.529e-36 5.529e-36 5.038e-32    1.0000
B            1 0.0094090 0.0094090   85.7312 1.503e-05 ***
C            1 0.0001440 0.0001440    1.3121    0.2851
A:B          1 4.516e-36 4.516e-36 4.115e-32    1.0000
A:C          1 1.176e-35 1.176e-35 1.072e-31    1.0000
B:C          1 0.0000810 0.0000810    0.7380    0.4153
Residuals    8 0.0008780 0.0001097
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Notice that there is no information on ABC as it is confounded.

Chapter 11

Two-Level Fractional-Factorial Designs

Most of the hard work in learning R has been done in the earlier chapters. Here we go through two examples from ED - the one on after-hours training and the magazine study. The former has replicates in its design, meaning that R is able to compute F-values for inference.

11.1 After-Hours Training Example

The data for this example is in the *eg11.7* file. In the data, a 0 indicates a low level and 1 a high. The design is 2^{3-1} half-replicate design. The confounded effect comes from the relation $I = ABC$.

```
> train <- read.table("eg11.7", header = T)
> summary(aov(resp ~ A * B * C, data = train))

            Df Sum Sq Mean Sq F value   Pr(>F)
A            1    294     294    14.7 0.001037 **
B            1     24      24     1.2 0.286338
C            1      6       6     0.3 0.589944
Residuals   20    400      20
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that R drops the two way interaction terms. Otherwise the table produced is identical to Table 11.21 in ED.

11.2 Magazine Ad Study Example

The data for this set is contained in *eg11.8*. The experiment is from a 2^{6-2} quarter-replicate design. The confounded effects are defined by the

$I = ABCD = ABEF = CDEF$ relation.

```
> adstud <- read.table("eg11.8", header = T)
> summary(aov(Y ~ A * B * C * D * E * F, data = adstud))
```

```
            Df Sum Sq Mean Sq
A            1     14      14
B            1   9264    9264
C            1     46      46
D            1  40100   40100
E            1 0.0625  0.0625
F            1 160601  160601
A:B          1      1       1
A:C          1      8       8
B:C          1      1       1
A:E          1 0.0625  0.0625
B:E          1     14      14
C:E          1 0.0625  0.0625
D:E          1      1       1
A:C:E        1     18      18
B:C:E        1      3       3
```

There are no F-values as the MSE cannot be estimated. Even though the entries in the table seem to be different from Table 11.27 given in ED, they are actually the same. It is just that R specifies different aliased effects from SPSS.