R Commands for –
Analysis of Variance, Design, and Regression:
Linear Modeling of Unbalanced Data

Ronald Christensen
Department of Mathematics and Statistics
University of New Mexico

This is a work in progress!

But it should be useful as is.

Preface

This is not a general introduction to programming in R! It is merely an introduction to generating the results in the book. As much as practicable, the chapters and sections of this guide give commands to generate results in the corresponding chapters and sections of the book. You should be able to copy the code given here and run it in R. An exception is that you will need to modify the locations associated with data files. Also, *if you are copying R code from a pdf file* into R, "tilde"

~

will often copy incorrectly so that **you may need to delete the copied version of tilde and retype it**. This may also be true for other characters like "caret" ^.

I learned how to run R by reading (and editing) Appendix C in Christensen et al. (2010). Two other tools that I have found very useful are Tom Short's R Reference card,
`http://cran.r-project.org/doc/contrib/Short-refcard.pdf`
and Robert I. Kabacoff's Quick-R website,
`http://www.statmethods.net/index.html`.
Given that I did not bother to learn R until the late oughts, it should not be surprising that programming is not my forte. I know lots of people who could create a much better introduction to programming in R than me, but there was no one else to perform this particular task.

As dismaying as I find the fact, it seems that relatively few students of statistics read books from beginning to end. Even I do not expect students to read a computing manual from beginning to end. As a result, I have made a positive effort to be repetitive between chapters about ideas that I think are particular important. *My ideal is that people would read the first three chapters and then skip around as needed. Chapter 3 contains the core ideas.*

Ronald Christensen
Albuquerque, New Mexico
March, 2015

Contents

Chapter 1

Introduction

1.1 Getting started

The first order of business is to download R. Vist the site http://www.r-project.org/ and follow the instructions.

When you open R, go to the File menu and open a “new script” window. Copy the scripts given here into the new window. To run part of the script, highlight the part you want to run, right click your mouse, and choose “Run line or selection.”

There is not a lot of computing associated with Chapter 1 of the book. This chapter introduces some elementary tools related to probability and graphing and some features of R that are useful.

1.2 Plots and probabilities

The code below was used to produce Figure 1.1. It begins by setting the values in the vector x as running from -3 to 3 at intervals 0.05 units apart. y uses `dnorm` to take on the values of the *normal density*. `dnorm`, get it? The three arguments in `dnorm` are: where to evaluate the normal density function, the mean, and the standard deviation (not the variance). The rest of the commands are used to set the labels and the vertical line for the plot.

```
x=seq(-3,3,.05)
y=dnorm(x,0,1)
y2=c(0,dnorm(1.6,0,1))
x2=c(1.6,1.6)
x1=c(-3,3)
y1=c(0,0)
plot(x,y,type="l",ylim=c(0,.4),ylab="",xlab="",at=c(0,1.6),labels=F)
mtext(expression(K(1-alpha)),line=2.5,side=1,at=1.6,padj=-1.25,cex=1.15)
mtext(expression(0),line=2.5,side=1,at=0,padj=-2.25,cex=1.15)
mtext(expression(0),line=2.5,side=2,at=0,padj=2.5,cex=1.1)
lines(x2,y2,type="l")
lines(x1,y1,type="l")
text(0,0.04,expression(1-alpha),lwd=2,cex=1.5)
text(1.9,0.03,expression(alpha),lwd=2,cex=1.5)
```

Figure 1.2 plots three curves at once. One is a normal density and the other two are $t(3)$ and $t(8)$ densities.

```
x=seq(-4,4,.05)
y=dnorm(x,0,1)
y1=dt(x,3)
y3=dt(x,8)
plot(x,y,type="l",ylim=c(0,.4),ylab="",xlab="",lty=1)
lines(x,y3,type="l",lty=3)
lines(x,y1,type="l",lty=2)
```

```
legend("topright",c("N(0,1)","t(8)","t(3)"),lty=c(1,3,2))
```

The purpose of including this code is twofold. First, it illustrates that other distributions, in this case the t distribution, work similarly to the normal, e.g. `dt(x,3)` evaluates the density of the $t(3)$ distribution. Second, it illustrates plotting three functions on the same graph.

Other plots in Chapter 1 use the commands `y=dchisq(x,8)` and `y=df(x,3,18)` to illustrate the densities of $\chi^2(8)$ and $F(3,18)$ distributions. To evaluate Bin(N,p) and Pois(λ) densities, use `dbinom(x,N,p)` and `dpois(x,lambda)`, respectively.

For a random variable y and real numbers u, $\Pr[y \le u] \equiv F(u)$ is known as the cumulative distribution function or cdf. If $y \sim N(\mu,\sigma^2)$, the cdf can be evaluated as `pnorm(u,mu,sigma)` where `p` stands for probability. Other distributions work similarly.

Figure 2.3 is very similar to Figure 1.1 except that it uses notation $t(1-\alpha,df)$ instead of $K(1-\alpha)$. $K(1-\alpha)$ is a generic term but $t(1-\alpha,df)$ can be computed as `qt(.95,df)` when, say, $1-\alpha = 0.95$. Here `q` stands for quantile. In general, the quantile associated with a probability α and a cdf F is the number u such that $\alpha = F(u)$. In the next chapter we will want to generate random observations from a normal distribution and we will see that it involves a minor modification of the distributional commands used here.

1.3 Reading data

You cannot analyze data in a computer package without somehow entering it. If the data are not numerous you might want to enter them by hand. For Example 2.1.1 in the book we might just use,

```
#  Enter data -- not from a data file
y=c(5, 22, 10, 12, 8, 17, 2, 25, 10, 10, 7, 7, 40, 7, 9, 17, 12,
12, 1, 13, 10, 13, 16, 3, 14, 17, 10, 10, 13, 59, 11, 13, 5, 12,
14, 3, 14, 15)
```

The easiest way to enter most data is to read it into the program. Virtually all of the data in the book is available on my website, http://www.stat.unm.edu/~fletcher/newavdr_data.zip. I think only one of the data files for the book includes labels.

Before reading the data, always look at the data file to identify what it is that you are reading. For the book, always compare the data file to the list of data in the book before proceeding. Below is an example of how to read the dropout rate data of Example 2.1.1 into an object called "drop". The first line includes the string of characters

```
"C:\\E-drive\\Books\\ANREG2\\newdata\\ex2-1-1.dat"
```

that specifies the location of the data file to be read. *This will change for every user.*

```
drop <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\ex2-1-1.dat",
  sep="",col.names=c("y"))
attach(drop)
drop
summary(drop)
```

Typing `drop` by itself lets you see what was read. Typing `summary(drop)` gives summary statistics. You will see these "read.table" commands in virtually every illustration.

My data files typically do not include column names/labels but if the data file does include column names, you can change the `read.table` command to

```
drop <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\ex2-1-1.dat",
  sep="",header=T)
attach(drop)
drop
summary(drop)
```

My data files are not in the *comma separated values* format but such files are easily read in R

```
read.csv("filename.csv")
```

1.4 Elementary transformations

The operators for addition, subtraction, multiplication, and division are `+ - * /`

Some transformations that you might find useful are

```
y=log(x)
y=exp(x)
y=sin(x)
y=cos(x)
y=x^(3)                                 # Or y=x^3 or y=x**3
y=asin(x)
```

The last transformation is the arcsine transformation whose argument x should be between 0 and 1. The penultimate transformation is $y = x^3$. (When copying from a pdf to R, the ˆ may need to be replaced.)

There is no serious programming in this volume. There is at least one simulation including a `for` loop listed in the index of www.stat.unm.edu/~fletcher/R-SL.

1.5 Housekeeping

To get help about a command in R just type `help(command)`.

In the previous section we presented

```
#  Enter data -- not from a data file
y=c(5, 22, 10, 12, 8, 17, 2, 25, 10, 10, 7, 7, 40, 7, 9, 17, 12,
12, 1, 13, 10, 13, 16, 3, 14, 17, 10, 10, 13, 59, 11, 13, 5, 12,
14, 3, 14, 15)
```

The symbol # allows us to place comments in an R program.

If you have a vector y with 50 observations and then want to do something else with y that involves only 25 observations you might want to clear the *entire* workspace so that you can get rid of R's expectation that y should have 50 observations. To do this, use

```
rm(list = ls())
```

After doing this, you have to start programming from scratch!

Suppose you have a variable y and want to define a new variable, say x, that is 2 times y. It is perhaps better programming practice to write

```
x <- 2 * y
```

but it is fewer keystrokes to write

```
x = 2*y
```

Both work.

1.6 Libraries/Packages

One advantage of learning R is that the most up-to-date statistical procedures tend to be made available first in R. A wide array of methodologies are available as R packages/libraries. In the next chapter we will use one for testing the normality of a random sample. To use a package, you need to install it (one time) on your computer. If you want to use the package, you need to call it every time you fire up R.

```
install.packages("nortest")  #Do this only once on your computer
library(nortest)  #Do this once every time you run R (and the package)
```

Other packages that we will use include: `car` (regression algorithms due to Fox, Weisberg, and associates), `leaps` (best subset regression), `lasso2` (lasso regression), `MASS` (algorithms due to Venables, Ripley, and associates), and `splines` (a version of nonparametric regression). By google-ing “r library(name)” you can access pdf files that both give information on the packages and give details on who was nice enough to provide these tools.

From November 8, 2018, to install a package I have had to run R as an administrator. To run as administrator I have to find R in the start menu, select the version that I want to run, right click to get options and go to the “More” option, which allows me to choose administrator. (You might be able to do this by right clicking the R icon.) Because this has become a pain, I have listed all of the libraries that are used in this book so they can be installed all at once. (I believe that some are pre-installed, so you don’t have to install them. You can check what is installed by going to the `load package...` menu option.)

```
install.packages("agricolae")
install.packages("car")
install.packages("cluster")
install.packages("lars")
install.packages("lasso2")
install.packages("leaps")
install.packages("lmtest")
install.packages("MASS")
install.packages("nortest")
install.packages("splines")
install.packages("splines2")
install.packages("stats")
install.packages("bestglm")
install.packages("StepReg")
install.packages("glmulti")
install.packages("lme4")
install.packages("lmerTest")
```

Information about all available packages is available by going to the R site: https://cran.r-project.org/web/packages/ I use the terms “package” and “library” interchangeably (outside of programming R).

1.6.1 Names

When running programs from packages/libraries or preprogrammed functions in R, often these programs create *objects* that have have *internal* structure that can be very useful to use. R uses a common structure for these internal objects that can be difficult to decipher. If a program produces an object that we identify as `object`, then the command `names(object)` will produce the names of the internal items contained in the object. To access one of these called, say, `item`, it will be accessed by the object name, a dollar sign, and then the item name, e.g.,

```
object = program()
names(object)
object$item
```

. For a specific illustration see Subsection 6.0.1.

Chapter 2

One-Sample

2.1 Introduction

In the previous chapter we showed how to read a sample of data. Adding one little command, `t.test`, computes most of what we need to generate the statistical results in the book, including, as illustrated below, a test of $H_0 : \mu = 10$ for the dropout rate data.

```
drop <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\ex2-1-1.dat",
  sep="",col.names=c("y"))
attach(drop)
drop
t.test(y,mu=10,conf.level = 0.95)
```

The output from `t.test` includes both the P value for the specific null hypothesis being tested and the confidence interval for the specific confidence level $1-\alpha$.

Although `t.test` is very useful for dealing with the data of Chapters 2 and 4, it is of little help outside those chapters. Most of this book focuses on a much more general R command for fitting linear models: `lm`. The next section shows how to generate the "t.test" output using the "lm" command that we will use in most of this book. Section 2.3 also shows how to incorporate missing data—something we want to do when deleting outliers.

2.2 Parametric inference

The basic form of the `lm` command is `lm(y ~ model)`. Models will be discussed in the next chapter. For now we just illustrate what needs to be done to get the results in the book. As in the book, results are presented in three subsections. Deleting the outliers will be handled in the next section.

2.2.1 Test $\mu = 10$

After entering the data,

```
dr <- lm((y-10) ~ 1)
drp=summary(dr)
drp
```

This provides an estimate that is $\bar{y}-10$, the appropriate standard error, t statistic, and P value. If we were testing $H_0 : \mu = 15$ we would use `(y-15)`.

Another less convenient way of doing parametric inference but a more convenient way to perform model testing uses the `offset` command that will be discussed more later.

```
x=y+1-y   # This is just a convenient way to create a vector of 1s
dr <- lm(y ~ offset(10*x))
drp=summary(dr)
drp
```

2.2.2 Confidence intervals

Computing confidence intervals requires a separate command.

```
dr <- lm(y ~ 1)
drp=summary(dr)
drp
confint(dr, level=0.95)

# Or you could compute the  confidence intervals from scratch.
R=drp$cov.unscaled
se <- sqrt(diag(R)) * drp$sigma
ci=c(dr$coef-qt(.975,drp$df[2])*se, dr$coef+qt(.975,drp$df[2])*se)
CI95 = matrix(ci,drp$df[1],2)
CI95
```

For a $1-a$ confidence interval, after defining a, replace .975 with $(1-a/2)$.

You can also get (very repetitive—a copy for every observation) confidence intervals by using the command `predict(dr,se.fit=T,interval="confidence",level=0.95)` after fitting the model. Deleting the outliers will be handled in the next section.

2.2.3 P values

Although the earlier test output contains a *P* value, you can compute a *P* value directly,

```
dr <- lm((y-10) ~ 1)
drp=summary(dr)
R=drp$cov.unscaled
se <- sqrt(diag(R)) * drp$sigma
PP=2*pt(-1*abs(dr$coef/se),drp$df[2])
PP
```

In fact, you could use a similar idea if you want to compute a *P* value directly from the computed value of t_{obs},

```
dr <- lm(y ~ 1)
drp=summary(dr)
R=drp$cov.unscaled
se <- sqrt(diag(R)) * drp$sigma
PP=2*pt(-1*abs((dr$coef-10)/se),drp$df[2])
PP
```

Here, `abs((dr$coef-10)/se)` is $|t_{obs}|$.

2.3 Prediction

We illustrate prediction using the drop out data with the two outliers deleted.

```
#  Enter data with outliers deleted.
y=c(5, 22, 10, 12, 8, 17, 2, 25, 10, 10, 7, 7, NA, 7, 9, 17, 12,
12, 1, 13, 10, 13, 16, 3, 14, 17, 10, 10, 13, NA, 11, 13, 5, 12,
14, 3, 14, 15)
dr <- lm(y ~ 1)
drp=summary(dr)
drp
#prediction
new = data.frame(x=c(1))
predict(lm(y~1),new,interval="prediction",level=0.95)
```

You can also get (very repetitive—a copy for every observation) prediction intervals by using the command
`predict(dr,se.fit=T,interval="prediction")`
after fitting the model.

2.4 Model testing

Model testing of $H_0 : \mu = 10$ for one sample uses two models including a reduced model that includes an `offset` and the `anova` command. By default, `lm` fits a constant term to every model. Subtracting 1 from a model, eliminates that default. More on this in Chapter 3.

```
full <- lm(y ~ 1)
x=y+1-y   # x is a vector of 1s.
red <- lm(y ~ offset(10*x)-1)
anova(full,red)
```

2.5 Normal plots

Normal plots occur frequently in the book. A computer program is necessary for finding the normal scores and **extremely** convenient for plotting the data and computing W'. The following commands provide a normal plot and the W' statistic for the complete dropout data y. As in the book, we illustrate deleting outliers and transforming the data. For the normal plot, the key item is `qqnorm`.

```
#  Enter data -- not from a data file
y=c(5, 22, 10, 12,  8, 17,  2, 25, 10, 10,  7,  7, 40,  7,  9, 17, 12, 12,
1, 13, 10, 13, 16,  3, 14, 17, 10, 10, 13, 59, 11, 13,  5, 12, 14,  3, 14,
15 )
qqnorm(y,ylab="Drop rates")

# Enter data with observations deleted.
yy=c(5, 22, 10, 12,  8, 17,  2, 25, 10, 10,  7,  7, NA,  7,  9, 17, 12, 12,
1, 13, 10, 13, 16,  3, 14, 17, 10, 10, 13, NA, 11, 13,  5, 12, 14,  3, 14,
15 )
qqnorm(yy,ylab="Drop rates")

#square root transformation of deleted data
yys=sqrt(yy)
qqnorm(yys,ylab="Drop rates")
```

The illustrations in the book of how normal plots should look used the command `rnorm(n,mu,sigma)` to generate a vector of length n containing random observations from a $N(\mu, \sigma^2)$.

2.5.1 *Shapiro-Francia test*

We can compute the Shapiro-Francia statistic to evaluate normality.

```
#Compute Shapiro Francia statistic for y.
x=qnorm(ppoints(y))
ys=sort(y)
Wprime=(cor(x,ys))**2
Wprime
```

Tables are provided in the book for determining if the statistic is significant.

It is also possible to install an R package for testing normality that performs the Shapiro-Francia test and other normality tests. The first step is to install the appropriate package. This is a one-time event for your computer. Simply type in `install.packages("nortest")`. After installing the package, before you can use it during an R session you need to invoke the package by typing

```
library(nortest)
```

You can then use the package to obtain the test via

```
sf.test(y)
```

There are some small differences in the computations. I suspect, but am not sure, that they are due to how the "rankits" are computed. More information about `nortest` is available at
http://cran.r-project.org/web/packages/nortest/index.html
One normality test that does not require installation of the package is `shapiro.test`.

I might add that for small sample sizes it is hard to tell if something is normal or not. And for large sample sizes, for most purposes other than prediction, it does not matter much if the data are normal.

2.6 Elementary transformations

Transformations were discussed in Chapter 1 of this guide. R commands for the three transformations discussed here and for the cubed root power transformation are given below. The cubed root is just to illustrate a general power transformation.

```
yl=log(y)
ys=sqrt(y)
yas=asin(sqrt(y))
ycr=y**(1/3)
```

The argument y for the arcsine transformation should be between 0 and 1.

2.7 Inference about σ

I do not know of any R program for doing inference on a single variance so the code below brute forces the confidence interval. By changing the dependent variable "yy" and its linear model, essentially the same code should work for any linear model.

```
# Enter data with observations deleted.
yy=c(5, 22, 10, 12,  8, 17,  2, 25, 10, 10,  7,  7, NA,  7,  9, 17, 12, 12,
1, 13, 10, 13, 16,  3, 14, 17, 10, 10, 13, NA, 11, 13,  5, 12, 14,  3, 14,
15 )

#set the alpha level as "a"
a=.05
dd<-lm(yy~1)
ddd=summary(dd)
ci=c((ddd$sigma**2)*ddd$df[2]/ qchisq(1-a/2,ddd$df[2]),
  (ddd$sigma**2)*ddd$df[2]/ qchisq(a/2,ddd$df[2]))
CIa = matrix(ci,1,2)
CIa
```

Chapter 3

Defining Linear Models

This chapter examines the syntax of R models from the most elementary models to the quite sophisticated. We begin with an two examples, to remind those users who are already familiar with the statistical concepts, of the syntaxes used to specify models in Minitab, R, and SAS. On a first reading of the manual, you can skip these two examples.

EXAMPLE 3.0.1. *Modeling Cheat Sheet*. We provide model syntax for models defined in Section 16.1 of the book. All three programs can fit the first form of the model. Minitab ONLY fits the first form. The R and SAS commands given below are for fitting the second form of the model.

$$\begin{aligned}[ABC] &\cong y_{ijkm} = G + A_i + B_j + C_k + [AB]_{ij} + [AC]_{ik} + [BC]_{jk} + [ABC]_{ijk} + e_{ijkm} \\ &\cong y_{ijkm} = [ABC]_{ijk} + e_{ijkm}.\end{aligned}$$

$$\begin{aligned}[AB][BC] &\cong y_{ijkm} = G + A_i + B_j + C_k + [AB]_{ij} + [BC]_{jk} + e_{ijkm} \\ &\cong y_{ijkm} = [AB]_{ij} + [BC]_{jk} + e_{ijkm}.\end{aligned}$$

$$\begin{aligned}[AB][C] &\cong y_{ijkm} = G + A_i + B_j + C_k + [AB]_{ij} + e_{ijkm} \\ &\cong y_{ijkm} = [AB]_{ij} + C_k + e_{ijkm}.\end{aligned}$$

$$\begin{aligned}[A_0][A_1][A_2][C] &\cong y_{ijkm} = G + A_{i0} + \gamma_1 x_j + \gamma_2 x_j^2 + A_{i1} x_j + A_{i2} x_j^2 + C_k + e_{ijkm}. \\ &\cong y_{ijkm} = A_{i0} + A_{i1} x_j + A_{i2} x_j^2 + C_k + e_{ijkm}.\end{aligned}$$

Model	Minitab	R	SAS
$[ABC]$	$A\|B\|C$	A:B:C-1	A*B*C / noint
$[AB][BC]$	$A\|B \quad B\|C$	A:B+B:C-1	A*B B*C / noint
$[AB][C]$	$A\|B \quad C$	A:B+C-1	A*B C / noint
$[A_0][A_1][A_2][C]$	$A\|X \quad A\|X2 \quad C$	A+A:X+A:X2+C-1	A A*X A*X2 C / noint

To fit different models, one needs to modify the part of the code that specifies the model. In Minitab's `glm`, models are usually specified in the `model` dialog box (or on the command line) and X and X2 have to be specified as covariates. In R, specifying models involves changes to, say, `lm(y ∼ A:B+C-1)` where A, B, and C all have to be prespecified as `factor` variables. In SAS's `proc glm`, modeling involves changes to `model y = A*B C/noint;` where A, B, and C all have to be prespecified as `class` variables.

I think the following statements are true. In R the model `A*B*C` is equivalent to `A+B+C+A:B+A:C+B:C+A:B:C`. In Minitab and SAS the model `A|B|C` is equivalent to `A B A*B C A*C B*C A*B*C`. □

Table 3.1: 2×3 *Factor Interactions*

a	A=factor(*a*)		*b*	B=factor(*b*)			A:B					
	a1	a2		b1	b2	b3	a1:b1	a1:b2	a1:b3	a2:b1	a2:b2	a2:b3
1	1	0	1	1	0	0	1	0	0	0	0	0
1	1	0	2	0	1	0	0	1	0	0	0	0
1	1	0	3	0	0	1	0	0	1	0	0	0
2	0	1	1	1	0	0	0	0	0	1	0	0
2	0	1	2	0	1	0	0	0	0	0	1	0
2	0	1	3	0	0	1	0	0	0	0	0	1
1	1	0	1	1	0	0	1	0	0	0	0	0
1	1	0	2	0	1	0	0	1	0	0	0	0
1	1	0	3	0	0	1	0	0	1	0	0	0
1	1	0	2	0	1	0	0	1	0	0	0	0
1	1	0	3	0	0	1	0	0	1	0	0	0
2	0	1	1	1	0	0	0	0	0	1	0	0
2	0	1	2	0	1	0	0	0	0	0	1	0
2	0	1	1	1	0	0	0	0	0	1	0	0
2	0	1	3	0	0	1	0	0	0	0	0	1

EXAMPLE 3.0.2. *Model Matrices Refresher*. This example illustrates the columns of the model matrix that correspond to various main effects and interactions in an unbalanced 2×3 two-factor ANOVA in which the second factor has quantitative levels of 1, 2, 3. *Model matrices are not introduced until Section 11.2,* so if you have not gotten that far in the book, skip this example.

A and *B* are factor variables and *b* has the quantitative levels for the second factor. In R define $bb \equiv b^2$. Using R modeling code on the left of the congruence symbol and with, on the right hand side, *b* known but A, B, γ, and (AB) denoting unknown parameters for the linear models:

$$\texttt{A+B - 1} \;\cong\; y_{ijk} = A_i + B_j + e_{ijk}$$

which is equivalent to

$$\texttt{A + b + bb - 1} \;\cong\; y_{ijkm} = A_{i0} + \gamma_1 b_j + \gamma_2 b_j^2 + e_{ijkm}.$$

This works with a quadratic function because the second factor has three levels and three points determine a quadratic function. If the second factor had 5 quantitative levels, we would need a fourth degree polynomial in *b* for it to be equivalent to the B_j effects. Similarly, when allowing interaction,

$$\texttt{A:B - 1} \;\cong\; y_{ijk} = (AB)_{ij} + e_{ijk}$$

which is equivalent to

$$\texttt{A + A:b + A:bb - 1} \;\cong\; y_{ijkm} = A_{i0} + A_{i1} b_j + A_{i2} b_j^2 + e_{ijkm}.$$

The model matrix columns that correspond to the factor terms and their interactions are given in Table 3.1. Table 3.2 replaces the B factor with the corresponding regression terms and interactions. The tables incorporate notation for the elementwise multiplication of vectors. For two vectors

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \quad \text{and} \quad w = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \quad \text{define} \quad v\!:\!w \equiv \begin{bmatrix} v_1 w_1 \\ \vdots \\ v_n w_n \end{bmatrix}.$$

□

This chapter describes general approaches to specifying fixed effect linear models in R. Chapter 3 in the book describes general approaches to statistical inference with Section 3.9 introducing

Table 3.2: 2×3 *Regression Interactions*

a	A=factor(*a*)		*b*	*bb*	A:*b*		A:*bb*	
	a1	a2			a1:*b*	a2:*b*	a1:*bb*	a2:*bb*
1	1	0	1	1	1	0	1	0
1	1	0	2	4	2	0	4	0
1	1	0	3	9	3	0	9	0
2	0	1	1	1	0	1	0	1
2	0	1	2	4	0	2	0	4
2	0	1	3	9	0	3	0	9
1	1	0	1	1	1	0	1	0
1	1	0	2	4	2	0	4	0
1	1	0	3	9	3	0	9	0
1	1	0	2	4	2	0	4	0
1	1	0	3	9	3	0	9	0
2	0	1	1	1	0	1	0	1
2	0	1	2	4	0	2	0	4
2	0	1	1	1	0	1	0	1
2	0	1	3	9	0	3	0	9

various linear models that are particularly useful. Most of this chapter is devoted to a discussion of how to specify those linear models in R. The chapter goes beyond those models because I think it is useful to consolidate in one place the fundamental ideas of specifying R models. It does not, however, discuss the random effects models that appear in Chapter 19.

We assume that y is a measurement random variable and that x is some predictor variable or that $x \equiv (x_1, \ldots, x_p)'$ is a vector of predictor variables. *In a computer file* all of the observations on y consist of a column of numbers and the x observations are either a single column of numbers or p different columns of numbers, one column for each component of the vector x. The components of the vector x can either be measurement (continuous) variables, categorical (factor, discrete) variables, or some combination of the two. We assumed in Section 3.9 of the book that

$$E(y) = m(x)$$

for some function m and described a number of different, commonly used, examples. When x contains only measurement variables, we construct regression models, when x contains only category variables, we construct ANOVA models, when x contains a combination of the two, we construct ACOVA models.

Of course we have to tell the computer program whether any component of the vector x is a measurement or category variable. Most computer programs have a default setting that, unless a variable is specified to be one thing, it is assumed to be the other. R assumes that all numeric variables are measurement variables, so category variables taking on numeric values have to be specified as such. Any variable that takes nonnumeric values is automatically taken as a factor.

Also, most computer programs for linear models default to include an intercept term (grand mean) in every model.

If you know how to define (linear) models in R, it is a simple matter to get R to perform a test on full and reduced models as in Section 3.1 of the book.

```
full <- lm(y ~ full_model)
red <- lm(y ~ reduced_model)
anova(red,full)
```

As in Subsection 3.1.1, when there is a biggest model and we want to do a test, use

```
big <- lm(y ~ big_model)
full <- lm(y ~ full_model)
red <- lm(y ~ reduced_model)
anova(red,full,big)
```

This actually reports two *F* statistics, we want the first.

Alternatively, the commands

```
fit <- lm(y ~ model)
fitp=summary(fit)
fitp
anova(fit)
```

provide both the table of coefficients and a table of sequential sums of squares and error from fitting linear models. Creating the standard three line ANOVA table in R seems to be a little awkward but can be constructed as in Chapter 11. R does not like to give the total sum of squares for ANOVA tables. Another way to get most of the three line ANOVA table is

```
full <- lm(y ~ model)
red <- lm(y ~ 1)
anova(red,full)
```

Another option is

```
anova(red,full,big, test="Cp")
```

although I have not actually seen the `test="Cp"` option produce any useful output. Other options for `test` are ”F”, ”Chisq”

The following are some relatively self-explanatory commands related to use of the “lm” command.

```
fit <- lm(y ~ model)       #fit the linear model and store results in "fit"
summary(fit)
coefficients(fit)                 #estimates of model parameters from "fit"
confint(fit, level=0.95)               #CIs for model parameters from "fit"
fitted(fit)                                   #predicted values from "fit"
residuals(fit)                                       #residuals from "fit"
df.residual(fit)                    #residual degrees of freedom from "fit"
length(y)                       #number of observations when none are missing
anova(fit)                                          #anova table for "fit"
predict(fit,new,interval="prediction",level=0.95)              #from "fit"
```

The second argument in “predict” is “new” which must contain the data for every variable in the model at which you want to predict. This can be a single point or a vector for every variable.

```
new = data.frame(first-x-variable=c(...),...,last-x-variable=c(...))
```

Here `first-x-variable` is replaced by whatever you called the first variable in your linear model, etc. If `new` is not specified, `predict` uses the data for the original fitting of the model.

The “aov” command can be used to replace the “lm” command. It seems to fit models the same way, it merely provides summary output that is a version of `anova(fit)` rather than outputing the table of coefficients. I believe that all of the commands listed above for `lm` also apply to `aov`. One advantage of `aov` is that it allows the specification of random effects, cf. Chapter 19.

3.1 One sample

If all the observations have the same mean value, you often have to use specialized software for this data structure, like R’s `t.test`. Often linear models programs do not allow fitting *only* a common mean to all observations. If a general program for linear models deals with the possibility, there is not much to specify. In R just define the model as

```
y ~ 1
```

Here the 1 indicates fitting a grand mean/intercept. In more complicated models, R assumes the existence of an intercept so you might think that specifying this 1 is redundant. But R does not seem to like a model without anything on the right of the tilde.

3.2 Two samples

For two samples, x should be a categorical variable with only two categories, specified by using

```
x = factor(x)
```

To follow the discussion in Chapter 4 of the book, I recommend fitting the model

```
y ~ x - 1
```

The `- 1` in the model tells it not to fit an intercept, so that the parameter estimates become the group sample means. You should compare this output to that obtained by fitting

```
y ~ x
```

in which the parameter estimates are one group sample mean and the difference between the sample means of the two groups.

It turns out that if all you want is a test of whether the group means are equal and if the two categories are coded as numbers, you do not need to specify that x defines categories. The usual summary output for fitting

```
y ~ x
```

still gives the test of whether the groups differ. This trick does not work if there are more than two categories!

Chapter 4 of the book also considers some extensions, i.e., paired comparisons and unequal variances. The `lm` command assumes equal variances, so it does not apply to the latter. As discussed in the next chapter, paired comparisons can be treated as either one-sample that consists of the differences between the pairs or as a two-way ANOVA without interaction.

3.3 Regression

In regression we assume that the vector x contains only measurement variables. In R, this is the default.

3.3.1 Simple linear regression

With a single predictor x, simple linear regression is

$$m(x) = \beta_0 + \beta_1 x$$

or

$$y_h = \beta_0 + \beta_1 x_h + \varepsilon_h, \qquad h = 1, \ldots, n.$$

It is modeled in R as

```
y ~ x
```

Notice that the intercept term is not specified and that this has the same structure as models used for two samples. The key point is that x is a measurement variable not a categorical variable. We will see later that this same R structure is used for one-way ANOVA when x is a factor variable with more than two categories.

If we wanted to force the regression through the origin the model becomes

$$y_h = \beta_1 x_h + \varepsilon_h, \qquad h = 1, \ldots, n.$$

and is modeled in R as

```
y ~ x - 1.
```

The model in question does not contain an intercept and the `- 1` is used to stop an intercept being fitted.

When x is a measurement variable, the two models

```
y ~ x - 1
y ~ x
```

are different models. It turns out that if x is a categorical variable,

```
y ~ x - 1
y ~ x
```

are equivalent models.

Incidentally, it is possible to subtract terms other than the intercept from models in R. However, until we introduce R conventions that automatically generate model terms, some of which we might not want, there is no point in adding a model term just to demonstrate that we can subtract it out again.

3.3.2 Polynomial regression

A cubic polynomial model

$$m(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

or

$$y_h = \beta_0 + \beta_1 x_h + \beta_2 x_h^2 + \beta_3 x_h^3 + \varepsilon_h, \qquad h = 1, \ldots, n,$$

is modeled in R as

```
y ~ x + I(x^2) + I(x^3)
```

Notice that the intercept term is not specified. Another way we could do this would be

```
x2 = x*x
x3 = x2*x
y ~ x + x2 + x3
```

One advantage of the former over the later is that to make predictions, the first form only requires us to specify the new value of `x`, whereas the second makes us specify all of `x`, `x2`, and `x3`.

Finally, R also has an option for fitting *orthogonal polynomials* of various degrees,

```
y ~ poly(x, degree = 3))
```

cf, Section 12.5. Fitting regular (nonorthogonal) polynomials is also an option,

```
y ~ poly(x, degree = 3, raw=TRUE)}
```

3.3.3 Multiple regression

Now suppose x is a vector of measurement variables with $p = 3$, i.e., $x = (x_1, x_2, x_3)'$. The multiple regression model is

$$m(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

or

$$y_h = \beta_0 + \beta_1 x_{h1} + \beta_2 x_{h2} + \beta_3 x_{h3} + \varepsilon_h, \qquad h = 1, \ldots, n.$$

Typically, when programming it is safer not to define variables using subscripts, so we write x_j as `xj` and the model is written in R as

```
y ~ x1 + x2 + x3
```

Notice that the intercept term is not specified.

3.3.4 Offsets

Suppose in the multiple regression model we knew that $\beta_2 = 5$ so that the model becomes

$$y_h = \beta_0 + \beta_1 x_{h1} + 5x_{h2} + \beta_3 x_{h3} + \varepsilon_h, \qquad h = 1, \ldots, n.$$

This can be modeled in R as

```
y ~ x1 + offset(5*x2) + x3
```

For a linear model like this, we can rewrite the linear model as

$$y_h - 5x_{h2} = \beta_0 + \beta_1 x_{h1} + \beta_3 x_{h3} + \varepsilon_h, \qquad h = 1, \ldots, n.$$

which can be modeled in R as

```
(y - 5*x2) ~ x1  + x3
```

Both of these techniques work for linear models. For nonlinear "generalized linear models," such as those treated near the end of the book, one must use the "offset" command.

3.4 ANOVA

We begin by assuming that x is a single category variable and then move on to vectors of category variables. If x is numerical, by default R assumes that x is a measurement variable, so we have to specify that x is categorical. When x is a vector, we have to specify that each numerical variable in the vector is categorical.

3.4.1 One-way ANOVA

A typical one-way ANOVA model is written

$$y_{ij} = \mu_i + \varepsilon_{ij}, \qquad i = 1, 2, \ldots, a, \; j = 1, \ldots, N_i.$$

In this context, the generic predictor variable x should really be thought of as i, the subscript in the model that identifies the groups. To specify that x is categorical, write

```
x = factor(x)
```

But just to avoid any possible confusion, I prefer to define a new version of x that is categorical,

```
xx = factor(x)
```

(In some contexts we might want to go back and forth between thinking of x as measurement or categorical in which case it is handy to have both versions.) The model is then specified as

```
y ~ xx - 1
```

The model in question does not contain an intercept with the `- 1` used to stop an intercept from being fitted.

An alternative form of the one-way model that contains a grand mean (intercept) is what most programs are designed to fit,

$$y_{ij} = \mu + \alpha_i + \varepsilon_{ij}, \qquad i = 1, 2, \ldots, a, \; j = 1, \ldots, N_i.$$

The model is written as

```
y ~ xx
```

For estimation R uses the side condition $\alpha_1 = 0$ which forces the estimate of μ to be the sample mean of group 1 and the estimate of α_j to be the difference between the sample mean of group j and the sample mean of group 1. As discussed in the book, other programs use other side conditions and it is important to be able to interpret the different output.

The one-way ANOVA with $a = 2$ model is identical to the model for two independent samples with equal variances and the model specification in R is the same.

3.4.2 Two-way ANOVA

Now suppose x is a vector of factor variables with $p = 2$, i.e., $x = (x_1, x_2)'$. As illustrated earlier, rewrite x_j as `xj`. Because `x1` and `x2` are factor variables, we specify

```
ii=factor(x1)
jj=factor(x2)
```

throughout this entire subsection.

3.4.2.1 Interaction

The model

$$y_{ijk} = \mu_{ij} + \varepsilon_{ijk}, \qquad i = 1,\ldots,a,\ j = 1,\ldots,b,\ k = 1,\ldots,N_{ij},$$

is written in R as

```
y ˜ ii : jj - 1
```

Remember that this is essentially a one-way ANOVA model!

Alternatively, the equivalent interaction model can be written

$$y_{ijk} = \mu + \alpha_i + \eta_j + (\alpha\eta)_{ij} + \varepsilon_{ijk}, \qquad i = 1,\ldots,a,\ j = 1,\ldots,b,\ k = 1,\ldots,N_{ij}.$$

and written in R as

```
y ˜ ii + jj + ii:jj
```

R, like many programs, has a shorter way to specify the model,

```
y ˜ ii * jj
```

When estimating parameters for this overspecified model, the following side conditions are used by R,

$$\alpha_1 = 0; \qquad \eta_1 = 0; \qquad (\alpha\eta)_{1j} = 0, \quad j = 1,\ldots,b; \qquad (\alpha\eta)_{i1} = 0 \quad i = 1,\ldots,a.$$

3.4.2.2 Additive effects

The additive effects model is

$$y_{ijk} = \mu + \alpha_i + \eta_j + \varepsilon_{ijk}, \qquad i = 1,\ldots,a,\ j = 1,\ldots,b,\ k = 1,\ldots,N_{ij}.$$

and is written in R as

```
y ˜ ii + jj
```

When estimating parameters, the following side conditions are used by R,

$$\alpha_1 = 0; \qquad \eta_1 = 0.$$

We mentioned earlier that we can also subtract effects. For example, to get the additive model we could specify the interaction model and then remove the interaction term,

```
y ˜ ii * jj - ii:jj
```

Similar models hold for $p > 2$ but get exponentially more complicated. The book examines in detail some cases with $p = 3$ and $p = 4$. Section 8 below also contains some discussion of higher-order models.

3.4.2.3 Sequential fitting

As discussed in the book, although the models themselves are equivalent, some computer output changes depending on whether one specifies the model

```
y ~ ii + jj + ii:jj
```

or

```
y ~ jj + ii + jj:ii
```

For example, to see different output try fitting, for some unbalanced data,

```
fit1 <- lm(y ~ ii + jj + ii:jj)
summary(fit1)
anova(fit1)
fit2 <- lm(y ~ jj + ii + jj:ii)
summary(fit2)
anova(fit2)
```

Although the end models are the same, the process of getting to the end model is different and the `anova` output contains information from the process in addition to information from the end result. For unbalanced data, you typically get different numbers from the `anova` command. For balanced data, only the order of presentation changes. Parameter estimation from `summary` depends only on the end model and does not change materially with different orderings of terms. In particular, when you ask R to fit

```
y ~ ii + jj + ii:jj
```

it sequentially fits the models

```
y ~ ii
y ~ ii + jj
y ~ ii + jj + ii:jj
```

R fits a sequence of models determined by the order in which you added terms of the larger model!

Incidentally, it is irrelevant in these models whether you specify `ii:jj` or `jj:ii`. However, the same is not true when using the `*` operator. In particular, `ii + jj + ii:jj` is equivalent to `ii*jj` whereas `jj + ii + jj:ii` is equivalent to `jj*ii`.

Somewhat more obscurely, in R `ii + jj + ii:jj` is equivalent to `ii:jj + ii + jj` whereas `jj + ii + ii:jj` is equivalent to `ii:jj + jj + ii`. My personal preference would have been for R to treat, say, `ii:jj + ii + jj` as equivalent to `ii:jj` because fitting an `ii` term after an `ii:jj` term actually contributes nothing. It seems that R is incorporating a convention that they think people will like but one that is at odds with the mathematics of linear model theory. Specifically, when you ask R to fit

```
y ~   ii:jj + ii + jj
```

it sequentially fits the models

```
y ~ ii
y ~ ii + jj
y ~ ii + jj + ii:jj
```

which is **not** the order of fitting that is suggested by the way in which terms were added to model.

Although this discussion has focused on the two-way with interaction model, issues of sequential fitting are pervasive with unbalanced data (including regression). This is not so much a problem as an opportunity. The extra computer output can save you from the chore of fitting some models. It never does any harm unless you misinterpret the results.

3.5 ACOVA and interaction

The following models were not discussed in Section 3.9 of the book but rather are discussed in Chapter 15. They involve a category variable x_1 written as `x1` and a measurement variable x_2 written as `x2` *or equivalently as* `z` (to make the notation more similar to the book). Throughout the section we assume

```
ii = factor(x1)
z=x2
```

3.5.1 ACOVA: parallel lines

The ACOVA model is

$$y_{ij} = \mu_i + \gamma z_{ij} + \varepsilon_{ij}, \qquad i = 1,2,\dots,a,\ j = 1,\dots,N_i.$$

written in R as

```
y ~ ii + z - 1
```

The a lines associated with the groups have intercepts μ_i and a common slope γ.

The alternative model

$$y_{ij} = \mu + \alpha_i + \gamma z_{ij} + \varepsilon_{ij}.$$

is written in R as

```
y ~ ii + z
```

When estimating parameters, the side condition $\alpha_1 = 0$ is used by R so that μ is the intercept of the line for the first group, α_i is the intercept of the ith group minus the intercept of the first, in other words, $\mu + \alpha_i$ is the intercept of the ith group, and again γ is the slope for all of the lines.

Note that this second R model is essentially the same form as that used for the two-way additive effects model in Subsubsection 3.4.2.2 except now one of the predictor variables is a measurement variable and the other is a factor variable.

3.5.2 Interaction: skew lines

We now consider ways of specifying lines for each group that can have different slopes as well as different intercepts. The simplest model to interpret is

$$y_{ij} = \mu_i + \gamma_i z_{ij} + \varepsilon_{ij}.$$

In this model μ_i is the intercept and γ_i is the slope for the line associated with the i group. The model in R is

```
y ~ ii + ii:z - 1
```

The version most similar to generalizing the traditional ACOVA model is

$$y_{ij} = \mu + \alpha_i + \gamma_i z_{ij} + \varepsilon_{ij}$$

in which the line for the ith group has intercept $\mu + \alpha_i$ and slope γ_i. Written in R this model is

```
y ~ ii + z:ii
```

and uses the side condition $\alpha_1 = 0$ for estimation. It does not matter if we write `z:ii` or `ii:z`.

Some folks like to specify a "hierarchical" version of the model (which I generally think is pretty silly but admittedly gives some useful computer output),

$$y_{ij} = \mu + \alpha_i + \beta z_{ij} + \gamma_i z_{ij} + \varepsilon_{ij}$$

written in R as

```
y ~ ii + z + ii:z
```

or

```
y ~ ii*z
```

For this model R uses $\alpha_1 = 0 = \gamma_1$. More on hierarchical models in Section 7.

3.6 Interaction in multiple regression

The multiple regression model with $p = 2$ and continuous predictors $x = (x_1, x_2)$ has

$$m(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

or

$$y_h = \beta_0 + \beta_1 x_{h1} + \beta_2 x_{h2} + \varepsilon_h, \qquad h = 1, \ldots, n.$$

This is an additive model in x_1 and x_2, cf. Section 9.9 of the book. The R model for this

```
y ~ x1 + x2
```

is essentially the same as that used for the two-way ANOVA additive effects model and the parallel lines ACOVA model except that now both of the predictor variables are measurement variables. Moreover, the effects of the two predictors simply add together.

A useful way of incorporating some interaction into the model is to fit

$$m(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2,$$

that is,

$$y_h = \beta_0 + \beta_1 x_{h1} + \beta_2 x_{h2} + \beta_3 x_{h1} x_{h2} + \varepsilon_h, \qquad h = 1, \ldots, n.$$

This is no longer an additive model in the original variables x_1 and x_2 (although it *is* an additive model in the variables x_1, x_2, $x_1 x_2$). The main reason for introducing this model here is to illustrate that the R model has essentially the same structure as the R models for two-way ANOVA with interaction and the ACOVA interaction model with skew lines. The R command is

```
y ~ x1 + x2 + x1:x2
```

or equivalently

```
y ~ x1*x2
y ~ (x1+x2)**2
```

With both variables being measurement variables there is yet another option for fitting the model

```
x1x2 <- x1*x2
y ~ x1 + x2 + x1x2
```

3.7 Hierarchical and nested models

The model

$$y_{ijk} = \mu + \alpha_i + \eta_j + (\alpha\eta)_{ij} + \varepsilon_{ijk}, \qquad i = 1, \ldots, a, \; j = 1, \ldots, b, \; k = 1, \ldots, N_{ij}.$$

is said to be hierarchical because the model includes not only the interaction terms $(\alpha\eta)_{ij}$ but also the lower order "main effect" terms α_i and η_j as well as the grand mean μ. A model is hierarchical whenever a model that includes an *m*th order interaction between some variables also includes all of the lower order interactions among those variables as well as all the main effects. Thus, if a model contains $(\alpha\eta\gamma)$ terms, it must also include $(\alpha\eta)$ terms, $(\alpha\gamma)$ terms, $(\eta\gamma)$ terms, and the main effect terms, the αs, ηs, and γs, as well as the grand mean μ. I have been at some pains in the book to point out that all of these lower order terms are meaningless, so I have little regard for this concept of a hierarchical model. (The term "hierarchical" takes on alternate meanings in alternate contexts, e.g., Bayesian statistics.) Some programs, like Minitab, insist that all models be hierarchical, using a similar concept of hierarchy for measurement variables. We have already seen that R uses the `*` operator to define hierarchical models. The model displayed earlier is most easily written as

```
y ~ x1*x2
```

I should point out that when using the rather artificial (but computationally very convenient) concept of interaction between continuous (measurement) variables, the lower order terms are **NOT** meaningless.

If you care about hierarchical models, you might care about nested models which are models that are not quite hierarchical. For example,

$$y_{ijk} = \mu + \alpha_i + (\alpha\eta)_{ij} + \varepsilon_{ijk}, \qquad i = 1,\ldots,a,\ j = 1,\ldots,b,\ k = 1,\ldots,N_{ij}$$

is said to have the $(\alpha\eta)_{ij}$ effects nested within the α_i effects. Again, fixed α_i effects are irrelevant in this model, so I find it hard to care about this concept of nesting. Nonetheless, R has specific code for defining nested models. With `x1=factor(x1)` and `x2=factor(x2)`, the model above could be written

```
y ~ x1/x2
```

which by definition is the same model as

```
y ~ x1 + x1:x2
```

Moreover, using R's operator for generating hierarchical models we can write the model as

```
y ~ x1*x2 - x2
```

Yes, as indicated earlier, R lets you subtract just about any term from a model.

3.8 Higher-order models

When discussing R, we will consider "higher-order" models to be those that involve $p > 2$, although the distinction is probably more common only when the number of factor variables is greater than 2. The reason we do this is because the methods in R for defining higher-order models are pretty much interchangeable regardless of whether the predictors are measurements or factors. When used on measurement variables, the R commands introduce interaction similar to Section 3.6 but do not introduce polynomial (power) terms.

For example, the code

```
y ~ (x1+x2+x3)^2
```

introduces second order interactions among the predictor variables and thus is identical to both of the models

```
y ~ (x1+x2+x3)*(x1+x2+x3)
y ~ x1*x2*x3 - x1:x2:x3
```

From the first two ways of writing the code you might be tempted to think that if all the predictors are measurements the model is

$$y_h = \sum_{r=0}^{2}\sum_{s=0}^{2}\sum_{t=0}^{2} \beta_{rst} x_{h1}^r x_{h2}^s x_{h3}^t + \varepsilon_h, \qquad h = 1,\ldots,n. \tag{3.8.1}$$

It is not! The model is actually

$$y_h = \beta_{000} + \beta_{100}x_{h1} + \beta_{010}x_{h2} + \beta_{001}x_{h3} + \beta_{110}x_{h1}x_{h2} + \beta_{101}x_{h1}x_{h3} + \beta_{011}x_{h2}x_{h3} + \varepsilon_h, \qquad h = 1,\ldots,n.$$

When all the predictors are factors with $x_1 \equiv i$, $x_2 \equiv j$, and $x_3 \equiv k$ the code means

$$y_{ijks} = \mu + \alpha_i + \eta_j + \gamma_k + (\alpha\eta)_{ij} + (\alpha\gamma)_{ik} + (\eta\gamma)_{jk} + \varepsilon_{ijks}, \qquad s = 1,\ldots,N_{hij}.$$

If only `x1` is a measurement variable, it means

$$y_{jks} = \mu + \beta_1 x_{jks1} + \eta_j + \gamma_k + \beta_{1j} x_{jks1} + \beta_{1k} x_{jks1} + (\eta\gamma)_{jk} + \varepsilon_{jks}, \qquad s = 1, \dots, N_{jk}$$

If both `x1` and `x2` are measurement variables, it means

$$y_{ks} = \beta_{00} + \beta_{10} x_{ks1} + \beta_{01} x_{ks2} + \gamma_k + \beta_{11} x_{ks1} x_{ks2} + \beta_{10k} x_{ks1} + \beta_{01k} x_{ks2} + \varepsilon_{jks}, \qquad s = 1, \dots, N_k$$

The models

```
y ~ (x1+x2+x3)^3
y ~ x1*x2*x3
```

are also identical.

To fit model (3.8.1) we can use the polynomial command

```
lm(y ~ poly(x1, x2, x3, degree = 2, raw=TRUE))
```

To fit corresponding orthogonal polynomials, remove the `raw` specification.

Similarly, we can fit *m*th order interaction models, where m is a positive integer no greater than p using, say, for $p = 4$,

```
y ~ (x1+x2+x3+x4)^m
```

With $p = 4$, the all measurement variable model is **not**

$$y_h = \sum_{r=0}^{m} \sum_{s=0}^{m} \sum_{t=0}^{m} \sum_{u=0}^{m} \beta_{rstu} x_{h1}^r x_{h2}^s x_{h3}^t x_{h4}^u + \varepsilon_h, \qquad h = 1, \dots, n.$$

The actual model for $m = 3$ and $p = 4$ has third order interactions and is

$$\begin{aligned} y_h = \; & y_h = \beta_{000} + \beta_{1000} x_{h1} + \beta_{0100} x_{h2} + \beta_{0010} x_{h3} + \beta_{0001} x_{h4} \\ & + \beta_{1100} x_{h1} x_{h2} + \beta_{1010} x_{h1} x_{h3} + \beta_{1001} x_{h1} x_{h4} + \beta_{0110} x_{h2} x_{h3} + \beta_{0101} x_{h2} x_{h4} + \beta_{0011} x_{h3} x_{h4} \\ & + \beta_{1110} x_{h1} x_{h2} x_{h3} + \beta_{1101} x_{h1} x_{h2} x_{h4} + \beta_{1011} x_{h1} x_{h3} x_{h4} + \beta_{0111} x_{h2} x_{h3} x_{h4} + \varepsilon_h, \qquad h = 1, \dots, n. \end{aligned}$$

The ANOVA model that use all factors is

$$\begin{aligned} y_{hijks} = \; & \mu + \alpha_h + \eta_i + \gamma_j + \delta_k \\ & + (\alpha\eta)_{hi} + (\alpha\gamma)_{hj} + (\alpha\delta)_{hk} + (\eta\gamma\delta)_{ij} + (\eta\delta)_{ik} + (\gamma\delta)_{jk} \\ & + (\alpha\eta\gamma)_{hij} + (\alpha\eta\delta)_{hik} + (\alpha\gamma\delta)_{hjk} + (\eta\gamma\delta)_{ijk} + \varepsilon_{hijks}, \qquad s = 1, \dots, N_{hijk}. \end{aligned}$$

I will leave it to you to worry about the various kinds of ACOVA models that mix measurements with factors.

Chapter 4

Two Samples

The command `t.test` performs most of the procedures discussed in Chapter 4. There are two ways to specify the two samples of data. You can either have one vector for each sample, say y_1 and y_2, or you can have one vector for all the data, say y, but also have an associated factor variable, say x, that identifies the sample for each observation. To use `lm` to analyze the data, the (x,y) data structure is needed.

4.1 Two correlated samples: paired comparisons

You need to know not only what sample an observation is in but also what pair it is in, so the (x,y) data structure is awkward for these problems. Using the y_1 and y_2 data structure, the vectors will have to have the same length and it is just assumed that the ith entry of y_1 is paired with the ith entry of y_2. One way to test equality of means is to use

```
t.test(y1,y2,paired=TRUE)
```

For the data in the book use

```
hard <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab4-1.dat",
  sep="",col.names=c("Pair","y1","y2"))
attach(hard)
hard
t.test(y1,y2,paired=TRUE,conf.level = 0.99)
```

Alternatively, you can fit a one-sample linear model to the differences between the pairs.

```
dr <- lm((y1-y2) ~ 1)
drp=summary(dr)
drp
anova(dr)
```

This allows prediction of the difference in a new pair as demonstrated in Chapter 2 and also predictions for new individual observations by treating the two samples separately as in Chapter 2.

Yet another way to use `t.test` on this problem is to compute the differences and then use `t.test` as a one-sample procedure.

```
d = (y1-y2)
t.test(d)
```

In Section 17.4.1 of the book we mention that the paired comparison is equivalent to fitting a randomized complete block model. To do this we need not only the (x,y) data structure but a third variable, say, z that identifies the pair. Both x and z need to be factor variables. Fit the two-way ANOVA

```
dr <- lm(y ~ x + z)
drp=summary(dr)
drp
anova(dr)
```

and look at the test associated with x to establish differences between samples. For Shewhart's hardness data, I do not have a data file set up for this analysis. **double check last line**

Paired comparisons are not to be confused with the *pairwise comparisons* that people make between different groups in analysis of variance.

4.2 Two independent samples with equal variances

Both data structures work fine for this problem, but if you have unequal group sizes with the (y_1, y_2) structure you might need to fill out the group having a smaller sample size with missing data symbols to make the data vectors the same length. Depending on your data structure, use one of the following commands.

```
t.test(y1,y2,var.equal = TRUE)
t.test(y~x,var.equal = TRUE)
```

Alternatively, with the (x,y) data structure you could use

```
fit <- lm(y ~ x)
fitp=summary(fit)
fitp
anova(fit)
```

It turns out that to test equality of the group means, when fitting an intercept, if x is a categorical variable with *only two numerical categories* you do not need to specify whether the variable is measurement or category. However, if you do not specify that x is a category, the estimates that `lm` gives you will depend on the way the categories have been coded—so if you want to do anything other than just test that the mean difference is 0, you will need to get creative.

For the final point total data of the book and a data file with the (x,y) structure use

```
pts <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab4-2.dat",
  sep="",col.names=c("x","y"))
attach(pts)
pts
summary(pts)
t.test(y ~ x,var.equal = TRUE)
```

or use `lm` specifying x as a factor.

```
pts <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab4-2.dat",
  sep="",col.names=c("x","y"))
attach(pts)
pts
xx = factor(x)
fin <- lm(y ~ xx)
finp=summary(fin)
finp
anova(fin)
```

Compare this to the results obtained when using `lm` without specifying x as a factor.

```
pts <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab4-2.dat",
  sep="",col.names=c("x","y"))
attach(pts)
pts
fin <- lm(y ~ x)
finp=summary(fin)
finp
anova(fin)
```

Below we demonstrate the (y_1, y_2) data structure.

```
pts <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab4-2a.dat",
  sep="",col.names=c("Obs","y1","y2"))
attach(pts)
pts
summary(pts)
t.test(y1,y2,var.equal = TRUE)
```

4.3 Two independent samples with unequal variances

Before using this procedure I urge you to read the discussion in the book that points out that testing for equality of means with unequal variances is not nearly as informative as testing for equality of means with equal variances. With equal variances and unequal means, one of the distributions is, in a very useful sense, larger than the other one. *Establishing that the means are different when the variances are different may not tell you much that is useful.*

As mentioned earlier, the `lm` command assumes equal variances so it is not applicable to these data. (`lm` can only handle unequal variances if you know the relative sizes of all the variances, something that rarely happens.) To handle unequal variances we use `t.test`. The default two-sample procedure in t.test has unequal variances, so one need not specify unequal variances. Depending on how the data were entered, use one of

```
t.test(y1,y2)
t.test(y~x)
```

The data file for the turtle data in the book has the (y_1, y_2) structure.

```
turt <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab4-3.dat",
  sep="",col.names=c("Index","y1","y2"))
attach(turt)
turt
yy1=log(y1)
yy2=log(y2)
t.test(yy1,yy2)
```

This only does the $\alpha = 0.01$ test from the book by giving a P value. The one-sample results are performed as in Chapter 2.

4.4 Testing equality of the variances

The R command `var.test` provides an F test for the equality of two variances. For the turtle data using the (y_1, y_2) data structure.

```
turt <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab4-3.dat",
  sep="",col.names=c("y1","y2"))
attach(turt)
turt
summary(turt)
yy1=log(y1)
yy2=log(y2)
var.test(yy2, yy1)
```

To demonstrate the use of the (x, y) data structure we use the Final Points data

```
pts <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab4-2.dat",
  sep="",col.names=c("x","y"))
attach(pts)
pts
```

```
xx = factor(x)
var.test(y~xx)
```

although this analysis was not in the book.

More generally, the structure of this command is

```
var.test(y1, y2, ratio = 1,
 alternative = c("two.sided", "less", "greater"),
 conf.level = 0.95, ...)
```

It can also be used to test whether the variances are the same in two completely different linear models. Below we illustrated that capability but only by specifying a one-sample (intercept only) model for each set of data.

```
var.test(lm(y1 ~ 1), lm(y2 ~ 1),conf.level = 0.95)
```

Chapter 5

Contingency Tables

If you are teaching a class on analyzing data, as opposed to a class on the use of linear models for analyzing unbalanced data, you should do some count data early on—which is what the book does.

The data in this chapter can be analyzed with the log-linear models of Chapter 21 but that level of sophistication seems premature. Therefore, we use specialized programs to look at these data, rather than the generalization of `lm` that is the `glm` command. In particular we use two rather similar commands: `prop.test` and `chisq.test`. The command `prop.test` only handles binary outcomes and gives confidence intervals but not Pearson residuals. `chisq.test` handles more general data, so does not give confidence intervals, but does give Pearson residuals.

The book does not discuss exact conditional tests (like Fisher's exact test) but `binom.test` makes such computations.

5.1 One binomial sample

The R code

```
prop.test(557,1835,correct=FALSE)
```

gives the confidence interval presented in the text. The R code for testing $H_0 : p = 1/3$ at $\alpha = 0.01$ is

```
prop.test(557,1835,p=.33333,conf.level = 0.99,correct=FALSE)
```

The code gives the P value for testing $H_0 : p = 1/3$ and the 99% confidence interval, which determines the $\alpha = 0.01$ level test. Unfortunately, `prop.test` only provides the square of the test statistic presented in the book.

5.1.1 Sign test

Obviously the whole point of the sign test is to turn a continuous data problem into a binomial problem. So with a little manipulation you can use `prop.test`. A google search turned up the fact that there is a package BSDA associated with a book by Larry Kitchens that includes a sign test function.

5.2 Two independent binomial samples

The data file contains three columns: admissions, rejections, and the total number of applicants. With this information, the simplest way to proceed is

```
berk <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-1.dat",
  sep="",col.names=c("Admit","Reject","Total"))
attach(berk)
berk
prop.test(Admit,Total,correct=FALSE)
```

The test statistic produced is the square of the test statistic in the book.

An alternative way to enter the data is to create a matrix of the admissions and rejections.

```
berk <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-1.dat",
  sep="",col.names=c("Admit","Reject","Total"))
attach(berk)
berk
summary(berk)
AD <- matrix(c(Admit,Reject),ncol=2)
AD
prop.test(AD,correct=FALSE)
```

We could replace `prop.test` with `chisq.test` (using the same arguments) and get the same test but slightly different output and options.

5.3 One multinomial sample

The following code gives the Pearson test, residuals, and null hypothesis expected values.

```
birth <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-2.dat",
  sep="",col.names=c("Fem"))
attach(birth)
birth
J=(Fem+1-Fem)  # J is a column of 1s
pp=J/12   # pp is a column of hypothesized probabilities for each month
fit <- chisq.test(Fem,p=pp,correct=FALSE)
fit
fit$expected
fit$residual
resid(fit)
```

This is not in any way binary data, so it is not surprising that I have not been able to get similar results out of `prop.test`.

The following commands might be useful for performing one of the exercises in the book.

```
pp=c(31,28,31,30,31,30,31,31,30,31,30,31)
chisq.test(Fem,p=pp,correct=FALSE,rescale.p=TRUE)
```

5.4 Two independent multinomial samples

The command `chisq.test` allows us to enter a matrix with the female and male births. Since that is precisely what is in the data file, we can read the data file (almost) directly into `chisq.test`. The procedure provides access to Pearson residuals and estimated expected values, things that `prop.test` does not give.

```
birth <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-3.dat",
  sep="",col.names=c("F","M"))
attach(birth)
birth
fit <- chisq.test(birth,correct=FALSE)
fit
fit$expected
fit$residual
```

You get the same test statistic and *P* value if you replace `chisq.test` with `prop.test`. Even though this is not binary data—12 categories, not 2 categories—the fact that we have exactly two samples lets us use `prop.test`.

Instead of creating a matrix of the female and male births, using `prop.test` we could compute the total number of births and use that together with either the female or the male births.

```
birth <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-3.dat",
  sep="",col.names=c("F","M"))
attach(birth)
birth
T <- F+M
prop.test(F,T,correct=FALSE)
```

Yet another way to enter the data is as a vector of birth counts with corresponding vectors that specify the month of birth and the sex of the child. This is a very convenient way to specify the data for using the modeling procedures in Chapter 21. To apply `chisq.test`, we first need to form a table out of the data using `xtabs`. In fact, if all we want is the Pearson test, we do not need to run `chisq.test` because `xtabs` will give the test as part of a `summary`. However, `chisq.test` also gives Pearson residuals and estimated expected values.

```
birth <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-3a.dat",
  sep="",col.names=c("O","Month","Sex"))
attach(birth)
birth
BR  <- xtabs(O~Month+Sex)
BR
summary(BR)
fit <- chisq.test(BR,correct=FALSE)
fit
fit$expected
fit$residual
```

5.5 Several independent multinomial samples

With the exception that `prop.test` no longer applies, this works pretty much as the previous section worked except now the data matrix will have more than two columns. The data file contains a list of counts along with indices for 4 occupations and 3 religions.

```
lazer <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-6.dat",
  sep="",col.names=c("O","Rel","Occ"))
attach(lazer)
lazer
laz <- xtabs(O~Rel+Occ)
laz
fit <- chisq.test(laz,correct=FALSE)
fit
fit$expected
fit$residual
```

If `tab5-6a.dat` existed (it does not) it would look just like Table 5.6 in the book, i.e., it would have four columns of numbers that we would label A, B, C, D and three rows of numbers, one for each religion. Such a data file could be read directly into `chisq.test`.

```
lazer <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-6a.dat",
  sep="",col.names=c("A","B","C","D"))
attach(lazer)
lazer
fit <- chisq.test(lazer,correct=FALSE)
fit
```

```
fit$expected
fit$residual
```

5.6 Lancaster–Irwin partitioning

To perform Lancaster-Irwin partitioning, you “need” to manipulate the data to create appropriate subtables. You can do that in your favorite editor. I might mention that in Chapter 21 we discuss performing Lancaster-Irwin partitioning by manipulating the subscripts used to define log-linear models.

Chapter 6

Simple Linear Regression

Chapter 6 of the book introduces the theory and applications of simple linear regression along with two “special cases.” The first special case has been alluded to earlier, the fact that in a two-sample test (cf. Chapter 4) you do not need to specify that the predictor variable is a factor in order to get the test for no differences between groups. The second “special case” is actually the generalization to multiple regression. We introduce multiple regression early so that we can exploit its mathematics and computing to look at models that treat the one variable used in simple linear regression in more sophisticated ways. Later, in Chapter 9, we begin a fuller treatment of multiple regression per se. For R commands on the special cases, see Chapters 4 and 9.

The following R script will give you most of what you need for simple linear regression. Indeed, with minor modifications it will give you most of the inferential statistics that you will need for fitting any linear model.

```
rm(list = ls())
# Read the data
coleman.slr <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab6-1.dat",
  sep="",col.names=c("School","x","y"))
attach(coleman.slr)
coleman.slr
#summary(coleman.slr)

# Table of coefficients and ANOVA table
cr <- lm(y ~ x)
crp=summary(cr)
crp
anova(cr)

# Confidence intervals
confint(cr, level=0.95)

# Line estimation and prediction at x=-16.04
new = data.frame(x=c(-16.04))
predict(cr,new,se.fit=T,interval="confidence")
predict(lm(y~x),new,interval="prediction")
# "cr" and "lm(y~x)" are interchangeable here.
```

The vast majority of the analyses we will run in this book can be computed by changing the (two lines of the) `read.table` *command to enter the appropriate data and changing the*

```
cr <- lm(y ~ x)
```

command to allow for fitting an appropriate model. The `new` object in the `predict` command also requires modification.

For simple linear regression, another useful tool is to plot the data with the fitted line superimposed.

```
plot(x,y)
abline(cr)
```

The next script produces items discussed in Chapter 7. The material is integral to any good data analysis, so I consider it useful to keep all of the commands in close proximity. As with the earlier commands, they should work for any linear model. (We frequently change the name of the `lm` output which requires corresponding changes in this code.) The script is repeated in the next chapter. The Chapter 7 material includes the production of four plots. If you run all of the `plot` commands at once, you will only see the last one, so the plots need to be run one at a time. It is possible to get them all to print out at once, and we also illustrate that. But if you run them all at once, the plots are much smaller.

```
# Read the data and run
cr <- lm(y ~ x)
crp=summary(cr)
# The code incorporates the names "cr" and "crp".

# Create a table of diagnostic statistics
infv = c(y,cr$fit,hatvalues(cr),rstandard(cr),
  rstudent(cr),cooks.distance(cr))
inf=matrix(infv,I(crp$df[1]+crp$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","r","t","C")))
inf
# Note: The computation assumes NO missing observations

# Normal and two residual plots:
# Run one plot at a time unless you know how to make matrix plots
qqnorm(rstandard(cr),ylab="Standardized residuals")

plot(cr$fit,rstandard(cr),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
plot(x,rstandard(cr),xlab="x",ylab="Standardized residuals",
main="Residual-Socio plot")

#leverage plot
Leverage=hatvalues(cr)
plot(School,Leverage,main="School-Leverage plot")

# Shapiro-Francia Statistic
rankit=qnorm(ppoints(rstandard(cr),a=I(3/8)))
ys=sort(rstandard(cr))
Wprime=(cor(rankit,ys))**2
Wprime
```

Or, for Shapiro-Francia, if you have installed the package `nortest` as in Subsection 2.5.1, use

```
library(nortest)
sf.test(rstandard(cr))
```

If you want to produce all four plots at once, use the following code to produce a 2×2 matrix of plots.

```
par(mfrow=c(2,2))
qqnorm(rstandard(cr),ylab="Standardized residuals")
```

```
plot(cr$fit,rstandard(cr),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
plot(x,rstandard(cr),xlab="x",ylab="Standardized residuals",
main="Residual-Socio plot")
Leverage=hatvalues(cr)
plot(School,Leverage,main="School-Leverage plot")
```

Once you run `par(mfrow=c(2,2))`, any plots you produce will appear in this matrix form. To stop this, run `par(mfrow=c(1,1))`.

6.0.1 Names

In the program above we occasionally used variables of the form

```
object$item
```

as we had earlier done in Chapter 8. Specifically, in this program we used

```
cr$fit
```

which gave us the fitted values. We used the `lm` program to construct the object `cr` and this is one of many variables that are *internal* constructs of the `cr` object. As another example, to find degrees of freedom we used *elements* of the

```
crp$df
```

variable where `crp` was the object we obtained from `summary(cr)`. The anova(cr) command also produces an object that has parts that we can use. How do we know what these internal variables are? To find out what internal items are available for our use within a particular object, enter `names(object)`. In this example that could be `names(cr)` or `names(crp)` or `names(anova(cr))`. In particular, these give

```
> names(cr)
 [1] "coefficients" "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"       "qr"            "df.residual"
 [9] "xlevels"       "call"         "terms"         "model"
> names(crp)
 [1] "call"          "terms"        "residuals"     "coefficients"
 [5] "aliased"       "sigma"        "df"            "r.squared"
 [9] "adj.r.squared" "fstatistic"   "cov.unscaled"
> names(anova(cr))
[1] "Df"      "Sum Sq"  "Mean Sq" "F value" "Pr(>F)"
```

Many of the longer names can be shortened.

To identify precisely what things are being produced, often it is useful to look at both the printout from the object and the printout of the internal variable. Check out, say,

```
crp
crp$df
cr.aov=anova(cr)
cr.aov
cr.aov$Df
cr.aov$Sum
cr.aov$Mean
```

The last two entries give variables containing the sums of squares and the mean squares, respectively, from the ANOVA table and illustrate that the names produced by `names` can be shortened. Also note that when using `lm`, the `df` item within the object `summary` is different from the `Df` item within object `anova`.

6.6 An alternative model

To create the alternative model read in the data and run

```
xc = x - mean(x)
cr <- lm(y ~ xc)
crp=summary(cr)
crp
anova(cr)
```

6.7 Correlation

We have already used the correlation command in R to obtain the Shapiro-Francia statistic. The command for computing the correlation between `x` and `y` is simply, `cor(x,y)`.

6.8 Two-sample problems

See Chapter 4.

6.9 Multiple Regression

See Chapter 9

Chapter 7

Model Checking

7.1 Recognizing randomness

The only computing was in producing plots. All of that information is elsewhere on my webpage.

7.2 Checking Assumptions

In addition to what is in *ANREG*, the package `library(lmtest)` includes versions of the Durbin-Watson test for serial correlation and the Breusch-Pagan/Cook-Weisberg test for heterogeneity. See *PA-V*, Chapter 12 (Chapter 13 in older editions) for discussion of these tests.

7.2.1 Another example: Hooker data

This is a second example of a simple linear regression, one in which the model does not fit the data. It is used to illustrate model checking devices. The following code is pretty much the same as that for the Coleman report data of the previous chapter with a few names changed.

```
hook <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab7-1.dat",
sep="",col.names=c("Case","Temp","Pres"))
attach(hook)
hook

hk <- lm(Pres ~ Temp)
hkp=summary(hk)
hkp
anova(hk)

confint(hk, level=0.95)

#ploting data with fitted line
plot(Temp,Pres)
abline(hk)                 # Or    abline(hk$coef)

infv = c(y,hk$fit,hatvalues(hk),rstandard(hk),rstudent(hk),
     cooks.distance(hk))
inf=matrix(infv,I(hkp$df[1]+hkp$df[2]),6,dimnames = list(NULL,
  c("y", "yhat", "lev","r","t","C")))
inf

qqnorm(rstandard(hk),ylab="Standardized residuals")
plot(hk$fit,rstandard(hk),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
```

```
plot(x,rstandard(hk),xlab="Temp",
ylab="Standardized residuals",main="Residual-Temp plot")

Case=seq(1:hkp$df[1]+hkp$df[2])
Leverage=hatvalues(hk)
plot(Case,Leverage,main="Case-Leverage plot")

# Wilk-Francia
rankit=qnorm(ppoints(rstandard(hk),a=I(3/8)))
ys=sort(rstandard(hk))
Wprime=(cor(rankit,ys))**2
Wprime
```

7.2.2 *Outliers: Coleman data*

The code was given in the previous subsection. Probabilites and percentiles (quantiles) of t distributions were discussed in Chapter 2. That is what is needed to perform the Bonferonni method. The `cars` package has automated the process with `outlierTest(fit)`

7.2.3 *Effects of high leverage*

I leave this to you.

7.3 Transformations

7.3.1 *Circle of transformations*

Code for transformations has been discussed in Chapters 1 and 2. To fit the log transformed hooker data, use

```
hook <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab7-1.dat",
sep="",col.names=c("Case","Temp","Pres"))
attach(hook)
hook
LPres = log(Pres)
hk <- lm(LPres ~ Temp)
hkp=summary(hk)
hkp
anova(hk)
```

and then proceed as in Subsection 7.2.1.

7.3.2 *Box-Cox transformations*

For the Hooker data, pick, say, $\lambda = 0.5$.

```
y=Pres
lambda = .5
ytilde = exp(mean(log(y)))                      #Geometric mean
zlambda = log(y) * ytilde
# The next line should only take effect if lambda is not equal to 0
# But it does not seem to
# Comment it out if you want to have lambda=0
zlambda = (y^lambda - 1)/(lambda*(ytilde^(lambda-1)))[lambda != 0]
hklambda <- lm(zlambda ~ Temp)
```

```
hklambdap=summary(hklambda)
hklambdap
anova(hklambda)
```

Do this for $\lambda = 0.5, 1/3, 0.25, 0, -0.25, -0.5$.

I have yet to actually run these programs. Both of the packages `MASS` and `car` have automated this procedure with `boxcox` and `boxCox`, respectively. According to the website Inside-R, the `car` "routine is an elaboration of the boxcox function in the MASS package." **I haven't tried these out.**

```
boxcox(model-or-fit,plotit=T)
boxcox(g,plotit=T,lambda=seq(-0.5,1.5,by=0.1))
```

7.3.3 Constructed variables

To check if Hooker's dependent variable needs transforming.

```
y = Pres
hk <- lm(y ~ Temp)
hkp=summary(hk)
hkp
anova(hk)

Lytilde = mean(log(y))     #Log of the geometric mean
w = y * (log(y)-Lytilde-1)
hkw <- lm(y ~ Temp + w)
summary(hkw)

wtilde = hk$fit * log(hk$fit)
hkwtilde <- lm(y ~ Temp + wtilde)
summary(hkwtilde
)

yhatsq = hk$fit^2
hkTukey <- lm(y ~ Temp + yhatsq)
summary(hkTukey)
```

To check if Hooker's predictor variable needs transforming, the Box-Tidwell procedure uses

```
y = Pres
x = Temp
xLx = x * log(x)
hkBT <- lm(y ~ x + xLx)
summary(hkBT)
```

7.4 Extras

I have not really checked these out.

In the previous chapter I included the diagnostic methods that I find most useful. Not surprisingly, R contains a variety of other methods that are preprogrammed, a few of which I will mention before going back to the specifics of the book.

```
fit <- lm(y ~ x)
# diagnostic plots
layout(matrix(c(1,2,3,4),2,2)) # optional 4 graphs/page
plot(fit)
```

```
influence(fit) # regression diagnostics

# If you have not done so already, run:   install.packages("car")
library(car)   # Companion to Applied Regression by Fox and Weisberg


outlierTest(fit) # Bonferonni p-value for most extreme obs
qqPlot(fit, main="QQ Plot") #qq plot for studentized resid
leveragePlots(fit) # leverage plots

# Influential Observations
# added variable plots
avPlots(fit)
# Cook's D plot
# identify D values > 4/(n-k-1)
cutoff <- 4/((nrow(mtcars)-length(fit$coefficients)-2))
plot(fit, which=4, cook.levels=cutoff)
# Influence Plot
influencePlot(fit, id.method="identify", main="Influence Plot",
sub="Circle size is proportial to Cook's Distance" )

ncvTest(fit)
# plot studentized residuals vs. fitted values
spreadLevelPlot(fit)

crPlots(fit)
# Ceres plots
ceresPlots(fit)
boxCox
boxTidwell
```

Chapter 8

Lack of Fit and Nonparametric Regression

8.1 Polynomial regression

Fit the fifth degree polynomial to the Hooker data.

```
hook <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab7-1.dat",
sep="",col.names=c("Case","Temp","Pres"))
attach(hook)
hook
summary(hook)
# Fit model (8.1.2)
x=Temp-mean(Temp)
x2=x*x
x3=x2*x
x4=x2*x2
x5=x3*x2
hk8 <- lm(Pres ~ x+x2+x3+x4+x5)
summary(hk8)
anova(hk8)
```

To get the lack-or-fit test, add

```
hk4 <- lm(Pres ~  x)
anova(hk4,hk8)
```

An alternative form of programming the polynomial model is

```
Pres ~ poly(Temp, degree = 5, raw=TRUE)}.
```

Yet another form, that makes prediction easy, is

```
hkp <- lm(Pres ~ Temp + I(Temp^2) + I(Temp^3) + I(Temp^4) + I(Temp^5))
hkpp <- summary(hkp)
hkpp
anova(hkp)
```

With this form, to make predictions one need only specify `Temp`, not all the powers of `Temp`.

Some computer programs don't like fitting the fifth degree polynomial on these data because the correlations are too high among the predictor variables. R is fine with it. If the correlations are too high, the first thing to do would be to subtract from the predictor variable its mean, e.g., replace `x` with `x - mean(x)`, prior to specifying the polynomial. If the correlations are still too high, you need to use *orthogonal polynomials*.

8.1.1 Picking a polynomial

The following list of commands (in addition to reading the data and defining the predictors as done earlier) give everything discussed in this subsection.

```
hk8 <- lm(Pres ~ x+x2+x3+x4+x5)
anova(hk8)
hk7 <- lm(Pres ~ x+x2+x3+x4)
summary(hk7)
anova(hk7,hk8)
```

While you already have everything for this subsection, you could finish fitting the entire hierarchy and ask to test them all.

```
hk6 <- lm(Pres ~ x+x2+x3)
hk5 <- lm(Pres ~  x+x2)
hk4 <- lm(Pres ~  x)
hk3 <- lm(Pres ~  1)
anova(hk3,hk4,hk5,hk6,hk7,hk8)
```

8.1.1.1 Orthogonal Polynomials

Orthogonal polynomials provide a way of getting the information for picking a polynomial in the form of useful t tests for all coefficients. Compare these results with our sequential fitting results. In particular, square the t tests and compare them to our F tests.

To fit orthogonal polynomials use

```
hko <- lm(Pres ~ poly(Temp, degree = 5))
summary(hko)
anova(hko)
```

The t tests from the orthogonal polynomials should be equivalent to the F tests given the command `anova(hk3,hk4,hk5,hk6,hk7,hk8)`.

8.1.2 Exploring the chosen polynomial

These are just the standard regression commands from Chapters 6 and 7 applied to the quadratic model.

```
hk <- lm(Pres ~ Temp + I(Temp^2))
hkp=summary(hk)
hkp
anova(hk)

# Diagonstic table (not given in book)
infv = c(Pres,hk$fit,hatvalues(hk),rstandard(hk),rstudent(hk),
     cooks.distance(hk))
inf=matrix(infv,I(hkp$df[1]+hkp$df[2]),6,dimnames = list(NULL,
  c("y", "yhat", "lev","r","t","C")))
inf

#Plots
plot(hk$fit,rstandard(hk),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
qqnorm(rstandard(hk),ylab="Standardized residuals")
# Wilk-Francia
rankit=qnorm(ppoints(rstandard(hk),a=I(3/8)))
ys=sort(rstandard(hk))
Wprime=(cor(rankit,ys))**2
Wprime
```

```
# More plots not given in book
plot(Temp,rstandard(hk),xlab="Temp",
ylab="Standardized residuals",main="Residual-Temp plot")
Leverage=hatvalues(hk)
plot(Case,Leverage,main="Case-Leverage plot")

#Predictions:  Note that with this model specification, only define Temp
new = data.frame(Temp=205)
predict(hk,new,se.fit=T,interval="confidence")
predict(hk,new,se.fit=T,interval="prediction")

#Lack-or-fit test, using output from previous subsection.
anova(hk5,hk8)       #Could also use anova(hk,hk8)
```

8.2 Polynomial regression and leverages

There are no new computing skills involved in this section.

8.3 Other basis functions

Most families of basis functions are defined on the unit interval. For the Hooker data we normalize the temperatures by subtracting the minimum value and dividing by the range to define a new predictor variable *x*.

```
hook <- read.table("c:\\E-drive\\Books\\ANREG2\\newdata\\tab7-1.dat",
sep="",col.names=c("Case","Temp","Pres"))
attach(hook)
hook
summary(hook)
#LPres=log(Pres)
range=30.5
min=180.5
x=(Temp-min)/range  # More generally x=(Temp-min(Temp))/range(Temp)
```

Although nonlinear, the Hooker data is pretty straight so we only use four basis functions

8.3.1 Sines and cosines

```
hk <- lm(Pres ~ Temp)
c1=cos(pi*1*x)
c2=cos(pi*2*x)
c3=cos(pi*3*x)
c4=cos(pi*4*x)
s1=sin(pi*1*x)
s2=sin(pi*2*x)
hook.sc <- lm(Pres ~ Temp + c1 + s1 + c2 + s2)
summary(hook.sc)
anova(hk,hook.sc)
hook.cos <- lm(Pres ~ Temp + c1 + c2 + c3 + c4)
summary(hook.cos)
anova(hk,hook.cos)
```

8.3.2 Haar wavelets

```
h1=1*as.logical(x<.25)
h2=1*as.logical(x>=.25 & x<.5)
h3=1*as.logical(x>=.5 & x<.75)
h4=1*as.logical(x>=.75)
hook.haar <- lm(Pres ~ Temp + h1 + h2 + h3 + h4)
summary(hook.haar)
hk <- lm(Pres ~ Temp)
anova(hk,hook.haar)
```

8.4 Partitioning methods

We give several methods of accomplishing the same thing. What is probably the best is saved for last.

```
hook <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab7-1.dat",
sep="",col.names=c("Case","Temp","Pres"))
attach(hook)
hook
summary(hook)

#Fit the two lines separately
hkl <- lm(Pres[Temp<191] ~ Temp[Temp<191])
summary(hkl)
anova(hkl)
hkh <- lm(Pres[Temp>=191] ~ Temp[Temp>=191])
summary(hkh)
anova(hkh)

#Fit the two lines at once
x=Temp
h <- Temp>=191       # h is an indicator of the temp being 191 or more
h2=1-h
x1=x*h
x2=x-x1
hkpt <- lm(Pres ~ h2+x2+h+x1-1)
hkpart=summary(hkpt)
hkpart

#Fit the two lines at once with low group as baseline
hkpt2 <- lm(Pres ~ x+h+x1)
hkpart2=summary(hkpt2)
hkpart2

#Fit two lines at once, Minitab-like
h3=h2-h
x3=x*h3
hkpt4 <- lm(Pres ~ x+h3+x3)
hkpart4=summary(hkpt4)
hkpart4
```

If you want to partition into 2 or more groups, the following is probably the easiest. Create a variable `h` that identifies the groups

```
hfac=factor(h)
hkpt3 <- lm(Pres ~ Temp +  hfac +  hfac:Temp)
hkpart3=summary(hkpt3)
hkpart3
anova(hkpt3)
```

In terms of sequential fitting, this fits the single line first and then fits lines to each partition set, so the sequential sums of squares for `hfac` and `hfac:Temp` go into the numerator of the *F* test.

If you have a more general model that also includes predictors `x1` and `x2`, you have a couple of choices. Either you can test for nonlinearity (i.e. lack of fit) in `Temp`, or you can test for lack of fit (nonlinearity) in the entire model. To test for nonlinearity in just temperature,

```
hfac=factor(h)
hkpt3 <- lm(Pres ~ x1 + x2 + Temp + hfac + hfac:Temp)
hkpart3=summary(hkpt3)
hkpart3
anova(hkpt3)
```

If you are doing this, you probably want the partition sets to only depend on `Temp`.

Alternatively, to test the entire model for lack of fit with an arbitrary collection of partition sets indexed by `h`,

```
hfac=factor(h)
hkpt3 <- lm(Pres ~ x1 + x2 + Temp + hfac + hfac:x1 + hfac:x2 + hfac:Temp)
hkpart3=summary(hkpt3)
hkpart3
anova(hkpt3)
```

Again, the sequential sums of squares are what you need to construct the *F*. Or you could get R to give you the test with

```
hfac=factor(h)
hkpt4 <- lm(Pres ~ x1 + x2 + Temp)
hkpt5 <- lm(Pres ~ hfac + hfac:x1 + hfac:x2 + hfac:Temp - 1)
anova(hkpt4,hkpt5)
```

8.4.1 Utts' method

It is possible to automate Utts' procedure, particularly by fitting the linear model, say,

```
rm(list = ls())
hook <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab7-1.dat",
sep="",col.names=c("Case","Temp","Pres"))
attach(hook)
hook
summary(hook)
hk <- lm(Pres ~ Temp)
```

and then defining an Utts subset of the data in terms of small values for `hatvalues(hk)`, for example,

```
k = .25
hkU <- lm(Pres[hatvalues(hk) < k] ~ Temp[hatvalues(hk) < k])
```

However, to reproduce what is in the book you would need to figure out the appropriate `k` values to obtain the 15 central points and the 6 central points. I created the subsets by manual inspection of `sort(hatvalues(hk))`. The commands

```
k = .05
hkU15 <- lm(Pres[hatvalues(hk) < k] ~ Temp[hatvalues(hk) < k])
```

```
summary(hkU15)
k = .035
hkU6 <- lm(Pres[hatvalues(hk) < k] ~ Temp[hatvalues(hk) < k])
summary(hkU6)
```

produced what I needed for the plots.

Assuming you know the number of data points in a *regression* `n`, Minitab would use `k = (1.1)*(n-df.residual(hk))/n`. When the dependent variable `y` has no missing observations, `n = length(y)`.

Below is the code for the two figures for this subsection.

```
#Utts1
xx=seq(185.6,197,.05)
yy1=-62.38+.42843*xx
plot(Temp,Pres,xlim=c(180.5,211),type="p")
lines(xx,yy1,type="l")

#Utts2
xx=seq(189.5,193.6,.05)
yy1=-48.123+.35398*xx
plot(Temp,Pres,xlim=c(180.5,211),type="p")
lines(xx,yy1,type="l")
```

The package `library(lmtest)` includes a version of Utts' Rainbow Test.

8.5 Splines

You would probably not want to do an entire spline fit without specialized software such as `library(splines)` and `library(splines2)`. We begin illustrating computing the two spline example from the book. We conclude with how to generalize that.

```
hook <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab7-1.dat",
sep="",col.names=c("Case","Temp","Pres"))
attach(hook)
hook

x=Temp
h <- Temp>=191
xsw = x*(1-h) + 191 * h       # The w in xsw is for "weird".
xs=(x-191)*h

# Nonstandard model derived from fitting separate lines.
hkpt <- lm(Pres ~ xsw+xs)
hkpart=summary(hkpt)
hkpart
anova(hkpt)

# Usual model derived from using first group as baseline.
hkpt2 <- lm(Pres ~ x+xs)
hkpart2=summary(hkpt2)
hkpart2
anova(hkpt2)

#lack-or-fit test
hk <- lm(Pres ~ x)
```

```
anova(hk,hkpt2)  # or anova(hk,hkpt)
```

To fit more than one linear spline term, say two, create two variables: `h1` that is 1 if you are greater than the first knot `k1` and 0 if you are less than the knot and also `h2` being 1 if you are greater than `k2` and 0 if you are less than the knot.

```
hook <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab7-1.dat",
sep="",col.names=c("Case","Temp","Pres"))
attach(hook)
hook

k1=185
k2=195
x=Temp
h1 <- Temp>=k1
xs1=(x-k1)*h1
h2 <- Temp>=k2
xs2=(x-k2)*h2
hkp <- lm(Pres ~ x+xs1+xs2)
hkpart=summary(hkpt2)
hkpart
anova(hkpt)
```

You should be able to fit a cubic spline with two interior knots using

```
hkp <- lm(Pres ~ x+I(x^2)+I(x^3)+I(xs1^3)+I(xs2^3))
```

8.6 Fisher's test

As is discussed in Chapter 12, this is just a test of the simple linear regression model against a one-way ANOVA.

```
hook <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab7-1.dat",
sep="",col.names=c("Case","Temp","Pres"))
attach(hook)
hook

y=Pres
x=Temp
fit <- lm(y ~ x)
grps = factor(x)
pe <- lm(Pres ~ grps)
anova(fit,pe)
```

If you have multiple predictors, say, `x1, x2, x3`, the procedure becomes

```
fit <- lm(y ~ x1+x2+x3)
xx1 = factor(x1)
xx2 = factor(x2)
xx3 = factor(x3)
pe <- lm(Pres ~ xx1:xx2:xx3)
anova(fit,pe)
```

Haven't run an example of the generalization.

Chapter 9

Multiple Regression and Diagnostics

As indicated before, the basic commands are extremely similar when moving from one measurement predictor to several of them.

9.1 Basic commands

```
coleman <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab6-4.dat",
sep="",col.names=c("School","x1","x2","x3","x4","x5","y"))
attach(coleman)
coleman

#Summary tables
co <- lm(y ~ x1+x2+x3+x4+x5)
cop=summary(co)
cop
anova(co)      # The default for anova is to give sequential sums of squares
#          The following device nearly prints out the three line ANOVA table
#                                               See Chapter 11 for details
Z <- model.matrix(co)[,-1]
co <- lm(y ~ Z)
anova(co)

confint(co, level=0.95)

#Predictions
new = data.frame(x1=2.07, x2=9.99,x3=-16.04,x4= 21.6, x5=5.17)
predict(lm(y~x1+x2+x3+x4+x5),new,se.fit=T,interval="confidence")
predict(lm(y~x1+x2+x3+x4+x5),new,interval="prediction")


infv = c(y,co$fit,hatvalues(co),rstandard(co),rstudent(co),
     cooks.distance(co))
inf=matrix(infv,I(cop$df[1]+cop$df[2]),6,dimnames = list(NULL,
  c("y", "yhat", "lev","r","t","C")))
inf

qqnorm(rstandard(co),ylab="Standardized residuals")

# Wilk-Francia
rankit=qnorm(ppoints(rstandard(co),a=I(3/8)))
ys=sort(rstandard(co))
```

```
Wprime=(cor(rankit,ys))**2
Wprime

plot(co$fit,rstandard(co),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
```

One of the first things we did in the chapter was print the correlations between y and the x_js. One can do that by running `cor(y,xj)` for $j = 1,\ldots,5$ but having created the model matrix `Z`, we can get them all by running `cor(y,Z)`. Similarly, the correlations given in Example 9.7.1 are produced by `cor(Z)`.

To delete cases 18 and 3 use the following code. Then rerun the earlier code except the diagnostic table needs to be modified.

```
y[18]=NA
y[3]=NA

infv = c(co$fit,hatvalues(co),rstandard(co),rstudent(co),
     cooks.distance(co))
inf=matrix(infv,I(cop$df[1]+cop$df[2]),5,dimnames = list(NULL,
  c( "yhat", "lev","r","t","C")))
inf
```

In Section 9.3 of the book we do two specific model comparison F tests. They can be generated using the following code.

```
cr <- lm(y ~ x1+x2+x3+x4+x5)
summary(cr)
anova(cr)

cr34 <- lm(y ~ x3+x4)
anova(cr34,cr)

cr134 <- lm(y ~ x1+x3+x4)
anova(cr34,cr134)
anova(cr34,cr134,cr)
```

The last call of `anova` gives the alternate version of the F test.

9.8 More on model testing

```
rm(list = ls())
chap <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\chapman.dat",
  sep="",col.names=c("Case","Ag","S","D","Ch","Ht","Wt","Coronary"))

attach(chap)
chap.mlr
#summary(chap)
ii=Case
x1=Ag
x2=S
x3=D
x4=Ch
x5=Ht
y=Wt
```

```
#Model (9.8.1)
m1=lm(y~x1+x2+x3+x4+x5)
summary(m1)
anova(m1)

#Model (9.8.2)
bsum=x2+x3
m2=lm(y~x1+bsum+x4+x5)
summary(m2)
anova(m2)

#Model (9.8.3)
bdif=x2-x3
m3=lm(y~x1+bdif+x4+x5)
summary(m3)
anova(m3)

#Model (9.8.4)
bsum=x2+x3
m4=lm(y~offset(.5*x3) + x1+bsum+x4+x5)
summary(m4)
anova(m4)

#Model (9.8.5)
bsum=x2+x3
m5=lm((y-.5*x3) ~ x1+bsum+x4+x5)
summary(m5)
anova(m5)

#Model (9.8.6)
m6=lm((y-3.5*x5) ~ x1+x2+x3+x4)
summary(m6)
anova(m6)



#Model (9.8.7)
bsum=x2+x3
m7=lm((y-.5*x3-3.5*x5) ~ x1+bsum+x4)
summary(m7)
anova(m7)

# or equivalently

m7=lm(y ~ offset(.5*x3+3.5*x5) + x1+bsum+x4)
summary(m7)
anova(m7)
```

9.10 Generalized additive models

I have not really checked out any of the rest of this chapter. This first thing looks especially questionable.

To fit the polynomial version of the book's model (9.10.2)

```
lm(y ~ poly(x1, degree = R, raw=TRUE) + poly(x2, degree = S, raw=TRUE))
```

To fit the polynomial version of the book's model (9.10.3) when $R = S$

```
lm(y ~ poly(x1, x2, degree = R, raw=TRUE))
```

Of course *R* and *S* need to be numbers, not just symbols.

9.10.1 Other useful tools?

```
plot(x3+x4~y,data=colman)
plot(x~y)
pairs(co)
```

Chapter 10

Diagnostics and Variable Selection

10.1 Diagnostics

Did it in the previous chapter.

10.2 Best subsets

The formula on page 244 of the text for Mallows's C_p is correct but for ANOVA models a better formula is

$$C_p = \frac{SSE(Red.)}{MSE(Full)} + n - 2\,dfE(Red.)$$

For models that are not regression models, it can be confusing to figure out the value of r used in the original formula. R likes to use AIC instead of Mallows's C_p, cf. Christensen (2020, Subsection 14.1.4). AIC differs by an additive constant from

$$n\log[SSE(Red.)] - 2\,dfE(Red.)$$

and the methods behave quite similarly.

To do a best subsect selection, start by running the full model.

```
rm(list = ls())
coleman <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab6-4.dat",
sep="",col.names=c("School","x1","x2","x3","x4","x5","y"))
attach(coleman)
coleman

#Summary tables
co <- lm(y ~ x1+x2+x3+x4+x5)
cop=summary(co)
cop
anova(co)
```

We will extract from this the model matrix in the next group of commands.

At some point you need to have run `install.packages("leaps")`. Then, to get the, say, `nb=3` best models for every number of predictors, run

```
# Best subset selection Table 10.11 in book.
library(leaps)
x <- model.matrix(co)[,-1]
# assign number of best models and number of predictor variables.
nb=3
xp=cop$df[1]-1
dfe=length(y)- 1- c(rep(1:(xp-1),each=nb),xp)
g <- regsubsets(x,y,nbest=nb)
```

```
gg = summary(g)
tt=c(gg$rsq,gg$adjr2,gg$cp,sqrt(gg$rss/dfe))
tt1=matrix(tt,nb*(xp-1)+1,4,
 dimnames = list(NULL,c("R2", "AdjR2", "Cp", "RootMSE")))
Cp.tab=data.frame(tt1,gg$outmat)
Cp.tab
```

Construction of the table uses the previously generated output from `co` and `cop`. `regsubsets` can also be used to perform stepwise regression as discussed in the next section. And `stepwise` from the next section can be used to perform best subset selection.

If you want to force some variables into all regressions, which would be reasonable if they had large *t* statistics in the full model, the following commands should work. You need to change`nfix` to be the number of variables being forced in. As before you need to determine the number of best models being fit, `nb`. `regsubsets` also needs you to specify the variables being forced in.

```
library(leaps)
x <- model.matrix(co)[,-1]
# assign number of best models and number of predictor variables.
nb=2
xp=cop$df[1]-1
nfix=2
dfe=length(y)- 1- nfix -c(rep((nfix+1):(xp-1),each=nb),xp)
g <- regsubsets(x,y,nbest=nb,force.in=c("x3","x4"))
gg = summary(g)
gg$outmat
tt=c(gg$rsq,gg$adjr2,gg$cp,sqrt(gg$rss/dfe))
tt1=matrix(tt,nb*(xp-nfix-1)+1,4,
 dimnames = list(NULL,c("R2", "AdjR2", "Cp", "RootMSE")))
Cp.tab=data.frame(tt1,gg$outmat)
Cp.tab
```

10.3 Stepwise methods

The package `StepReg` has a program `stepwise` allows `selection = "backward"` or `"forward"` or `"bidirection"`. The α to enter is specified by `sle =` α and the α to delete a term is specified by `sls =` α. It also allows searching for models with the lowest information criterion. It acts on the columns of the model matrix, so may do strange things with ANOVA models. Best subset selection is available using `selection = score`. **I have not yet run `stepwise`.**

R's `step` command is useful in many places, not just with `lm` and is designed to give reasonable answers even for ANOVA type models. It chooses models based on the AIC criterion. The 2020 edition of *Plane Answers* contains a discussion of AIC. A similar option is `stepAIC` from the *MASS* library.

First we fit the full model and then request the stepwise procedure.

```
rm(list = ls())
coleman <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab6-4.dat",
sep="",col.names=c("School","x1","x2","x3","x4","x5","y"))
attach(coleman)
coleman

#Summary tables
co <- lm(y ~ x1+x2+x3+x4+x5)
cop=summary(co)
cop
```

```
anova(co)


# This is the default backward elimination procedure.
co1 <- step(co, direction="backward")
cop1=summary(co1)
cop1
anova(co1)

# Forward selection is a little more complicated.
null = lm(y~1)
step(null, scope=list(lower=null, upper=co),direction="forward")
```

Actual stepwise uses `direction="both"`. There is an option `test="F"` to print out *F* statistics but it seems that the procedure still stops based on minimizing AIC. You could perhaps adjust the `k=` option to go past your stopping point and then use the *F* statistics to determine your stopping point.

`regsubsets` from the previous section can also be used to perform stepwise regression. It provides more options on rules for dropping and adding variables.

If you fit the full model to find MSE(F) then using `scale=MSE(F)` may cause use of the C_p statistic. **I haven't tried this yet.** Would also apply to `dropterm` and `addterm`.

I would guess that when `sls` is specified, telling it to use significance levels,`select="SL"`, would be redundant but I haven't checked.

10.4 Model selection and case deletion

No new computing here.

10.5 Lasso

```
coleman <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab6-4.dat",
sep="",col.names=c("School","x1","x2","x3","x4","x5","y"))
attach(coleman)
coleman
summary(coleman)

#Summary tables
co <- lm(y ~ x1+x2+x3+x4+x5)
cop=summary(co)
cop
anova(co)

x <- model.matrix(co)[,-1]


#At some point you need to have run  install.packages("lasso2")
library(lasso2)
tib <- l1ce(y ~ x1+x2+x3+x4+x5,data=coleman,bound = 0.56348)
tib


tib <- l1ce(y ~ x1+x2+x3+x4+x5,data=coleman,bound = 0.5+(seq(1:10)/100))
```

```
tib
```

It is most natural to apply the lasso (and ridge regression) to predictor variables that have been standardized by subtracting their mean and dividing by their standard deviation. If we create a matrix of the predictor variables, R has a command `scale` that does that.

```
coleman <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab6-4.dat",
sep="",col.names=c("School","x1","x2","x3","x4","x5","y"))
attach(coleman)
coleman
X = coleman[,2:6]
Xs = scale(X)
```

Or you could do the same thing by brute force using the matrix commands of Chapter 11.

```
X=matrix(c(x1-mean(x1),x2-mean(x2),x3-mean(x3),x4-mean(x4),x5-mean(x5),ncol=5)
Xs = X %*% diag(c(sd(x1),sd(x2),sd(x3),sd(x4),sd(x5))^(-1))
Xs
```

If you want, you could only center the variables or rescale them without centering them.

```
scale(coleman[,2:6], center = TRUE, scale = TRUE)
```

10.5.0.1 Pollution data

```
rm(list = ls())
coleman <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab9-4.dat",
 sep="",col.names=c("x1","x2","x3","a1","x4","x5","x6","x7","x8","x9","a2","a3","a4"
attach(coleman)
coleman
summary(coleman)

#Summary tables
co <- lm(y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9+x10)
cop=summary(co)
cop
anova(co)

x <- model.matrix(co)[,-1]


#At some point you need to have run  install.packages("lasso2")
library(lasso2)

tib <- l1ce(y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=coleman,standardize=FALSE,bound
tib
```

10.5.1 Cross-validation for tuning parameter

The following code originally written by my colleague **Yan Lu** includes options for cross-validatory selection of the lasso bound. It uses the `lars` package.

```
##Lasso for coleman data
install.packages("lasso2")
install.packages("lars")
library(lasso2)
library(lars)
```

```
coleman<- read.table(file="C:\\E-drive\\Books\\ANREG2\\newdata\\TAB6-4.DAT",
  sep="",col.names=c("School","x1","x2","x3","x4","x5","y"))
X <- as.matrix(coleman[,c(2,3,4,5,6)])
y<-coleman$y

# Run cross-validated lasso out of lars.
ans.cv <- cv.lars(X, y,type="lasso",K=10,index=seq(from = 0, to = 1, length =20))
# type=lasso is the default,
# index=seq(from = 0, to = 1, length =100) is the default
# K=10 K-fold cross validation is the default
# cv.lars produces the CV curve at each value of index


seid0 <- order(ans.cv$cv)[1] #id number for the minumum CV MSE
lambdamin <- ans.cv$index[seid0] #find \lambda that is associated with minimum CV
lambdamin #0.6842
minPLUSoneSE <- ans.cv$cv[seid0] +ans.cv$cv.error[seid0] # one SE from the minimum C
abline(minPLUSoneSE,0,lty=2) #looks like \lambda=0.45 is within the one SE from the
# last command adds a horizontal line to plot produced by cv.lars command

lasso3 <- l1ce(y~x1+x2+x3+x4+x5, data=coleman, bound=lambdamin)
coef(lasso3)
#(Intercept) x1 x2 x3 x4 x5
#15.0571948207 -1.1061254764 0.0008558439 0.5357023738 0.8508239046 0.0000000000
```

Chapter 11

Multiple Regression: Matrix Formulation

In this chapter we use R's matrix methods to produce results for multiple regression. At the end of the chapter we explore principal component regression and also address the most complicated ideas from Appendix A: eigenvalues and eigenvectors.

We produce the table of coefficients, the three line ANOVA table, and some diagnostic statistics. The commands are divided into sections but the sections depend on results computed in previous sections. **The methods used here are not good computational procedures**. Good programs for regression, or more general linear models, use more sophisticated methods of computation.

To produce multiple regression results, we use a collection of matrix commands available in R.

```
# General matrix commands
X <- model.matrix(co)                   # Model matrix of fitted model "co"
Z <- model.matrix(co)[,-1]              # Model matrix without intercept
t(X)                                    # transpose of X
diag(v,nrow=length(v))                  # diag. matrix from vector v
diag(A)                                 # diag. vector from matrix A
sum(diag(A))                            # trace of matrix A
%*%                                     # matrix multiplication
solve(A,b)                              # solves A %*% x = b for x
solve(A)                                # matrix inverse of A
rowsum(X)                               # sum of rows for a matrix-like object
rowSums(X)                              # This is a faster version
colsum(X)
colSums(X)
rowMeans(X)
colMeans(X)
scale(X,center=TRUE,scale=TRUE) # subtract col means, divide by col std. dev
rankMatrix(X)                           # Compute r(X). Can be dicey.
outer(x,y)                              # outer product:  xy'
inprod(x,y)                             # inner product:  x'y
```

The first order of business is constructing the model matrix from the data read in and showing that it is the same as that used by `lm`.

```
coleman <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab6-4.dat",
sep="",col.names=c("School","x1","x2","x3","x4","x5","y"))
attach(coleman)
coleman

# Create J, a column of ones.
J=x1+1-x1
# Create the model matrix from variables read.
X=matrix(c(J,x1,x2,x3,x4,x5),ncol=6)
X
```

```
# Extract the model matrix from lm.
co <- lm(y ~ x1+x2+x3+x4+x5)
XX=model.matrix(co)
XX
```

Note that X and XX are the same.

Now we compute the standard statistics. First we get the estimated regression parameters, then fill out the ANOVA table, finally we get the standard errors and t statistics. In the process we find the perpendicular projection operator (ppo) but in practice the ppo is an unwieldy creature to have saved in a computer's memory. For example, if $n = 1000$, M contains a million numbers.

```
# Find the vector of least squares estimates
Bhat=(solve(t(X)%*%X))%*%t(X)%*%y   #library(MASS) has a more general method
Bhat                                #       Bhat=(ginv(t(X)%*%X))%*%t(X)%*%y

#Find the perpendicular projection operator
M = X%*%solve(t(X)%*%X)%*%t(X)      #          M=X%*%(ginv(t(X)%*%X))%*%t(X)

# Compute the ANOVA table statistics.
yhat = M%*%y                   #This is more efficiently computed as X%*%Bhat
ehat = y - yhat
SSE = t(ehat)%*% ehat
n = length(y)
p = ncol(X)                                                 #  p = rankMatrix(X)
dfE <- n - p                     #Computing the rank of a matrix can be dicey
MSE = SSE/dfE
MSE = as.numeric(MSE)      #I got an error for Cov[Bhat] if I didn't do this
SSReg = t(yhat)%*%yhat - n*mean(y)**2         # Or  t(y)%*%yhat - n*mean(y)**2
dfReg = p-1
MSReg = SSReg/dfReg
SSTot = t(y)%*%y - n*mean(y)**2
dfTot = n-1
MSTot = SSTot/n-1
AOVTable = matrix(c(dfReg,dfE,dfTot,SSReg,SSE,SSTot,MSReg,MSE,MSTot),ncol=3)
AOVTable
co <- lm(y~X[,-1])
anova(co)
```

For comparison I included commands to print (most of) a three line ANOVA table using R's command anova. Normally anova does not print out a three line ANOVA table but it comes close if you define the model using a matrix (without a column of 1s) instead of using the modeling options discussed in Chapter 3.

Now construct the covariance matrix of $\hat{\beta}$ and compare it to the one given by lm before using it to obtain the standard errors and t statistics.

```
Cov = MSE*(solve(t(X)%*%X))
Cov
vcov(co)                      # lm's covariance matrix for model parameters
SE = sqrt(diag(Cov))
TabCoef=matrix(c(Bhat,SE,Bhat/SE),ncol=3)
TabCoef
cop = summary(co)
cop
```

Again, for comparison, I included commands to print lm's table of coefficients. I also needed to save the summary of the fit because I use it in the next series of commands.

The book discusses how to compute two of our standard diagnostic quantities: the leverages and the standardized residuals. Christensen (2011, Chapter 13) discusses computing diagnostic statistics. In particular, from the fit summary results “cop”, the leverages, and the standardized residuals we can compute *t* residuals and Cook’s distance. The leverages are just the diagonal elements of the ppo.

```
lev = diag(M)      #Computing M just to get the diagonal elements is overkill
sr = ehat/sqrt(MSE*(1-lev))                             #Standardized residuals
tresid=sr*sqrt((cop$df[2]-1)/(cop$df[2]-sr^2))                      # t residuals
C = sr^2 *lev/(p*(1-lev))                                        #Cook's distance
infhomemade=matrix(c(lev,sr,tresid,C),ncol=4)
infhomemade
infv = c(co$fit,hatvalues(co),rstandard(co),rstudent(co),
     cooks.distance(co))
inf=matrix(infv,I(cop$df[1]+cop$df[2]),5,dimnames = list(NULL,
  c( "yhat", "lev","r","t","C")))
inf
```

Again, for comparison, I have included our standard table of diagnostic values that lets `lm` do the heavy lifting of computations.

11.6 Principal component regression

Principal component regression using `prcomp`. The following commands reproduce the results in the book.

```
coleman <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab6-4.dat",
sep="",col.names=c("School","x1","x2","x3","x4","x5","y"))
attach(coleman)
coleman

fit <- prcomp(~ x1+x2+x3+x4+x5, scale=TRUE)
(fit$sdev)**2
summary(fit)
fit$rotation
co <- lm(y~fit$x)
cop <- summary(co)
cop
anova(co)
```

The book also gives the regression coefficients based on PC1, PC3, PC4 transformed back to the scale of the x_js.

```
gam <- c(co$coef[2],0,co$coef[4],co$coef[5],0)  #Zero coef.s for PC2 and PC5
int = mean(y)-(t(gam)%*% t(fit$rotation)) %*%(fit$center*((fit$scale)^(-1)))
PCbeta =c(int,(t(gam)%*% t(fit$rotation))*((fit$scale)^(-1)))
PCbeta
```

The is a section on eigenvalues and eigenvectors near the end of the book.

Chapter 12

One-Way ANOVA

This is pretty self-explanatory.

12.1 Suicide data

```
suic.aov <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab12-1.dat",
sep="",col.names=c("Age","group"))
attach(suic.aov)
suic.aov

#Summary tables
G=factor(group)
la=log(Age)
cr <- lm(Age ~ G-1)
crp=summary(cr)
crp
anova(cr)

confint(cr, level=0.95)

infv = c(la,cr$fit,hatvalues(cr),rstandard(cr),rstudent(cr),
     cooks.distance(cr))
inf=matrix(infv,I(crp$df[1]+crp$df[2]),6,dimnames = list(NULL,
  c("y", "yhat", "lev","r","t","C")))
inf
# delete y from table if missing observations

qqnorm(rstandard(cr),ylab="Standardized residuals")

plot(cr$fit,rstandard(cr),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")

plot(group,rstandard(cr),xlab="Group",
ylab="Standardized residuals",main="Residual-Group plot")


# Wilk-Francia
rankit=qnorm(ppoints(rstandard(cr),a=I(3/8)))
ys=sort(rstandard(cr))
Wprime=(cor(rankit,ys))**2
Wprime
```

12.2 Regression analysis of ANOVA data

```
suic <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab12-1.dat",
sep="",col.names=c("A","R"))
attach(suic)
suic
summary(suic)

#Summary tables
R=factor(R)
LA=log(A)
# Fit a one-way without a grand mean
sabc <- lm(LA ~ R - 1)
coef=summary(sabc)
coef
anova(sabc)
sabc1 <- lm(LA ~ R)
coef=summary(sabc)
coef
anova(sabc1)


# Tw0 ways to make indicator variables.
# Take indicators from model matrix
X = model.matrix(sabc)
# or X=model.matrix(~ R-1)
r1 = x[,1]
r2 = x[,2]
r3 = x[,3]
matrix(c(R,r1,r2,r3),ncol=4)
# The follow works if codes in R are 1,2,3
r1 = gsub("[^1]","0",R)
r2 = gsub("[^2]","0",R)
r3 = gsub("[^3]","0",R)
r1 = as.numeric(r1)
r2 = as.numeric(r2)/2
r3 = as.numeric(r3)/3
matrix(c(R,r1,r2,r3),ncol=4)
```

The `matrix` commands simply allow one to see what has been created.

12.3 Modeling contrasts

```
mand <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab12-4.dat",
sep="",col.names=c("yy","C2","C3","C4","C5","C6","C7","C8","C9"))
attach(mand)
mand

ii = factor(C2)
y = log(yy)
fit <- lm(y ~ ii)

C3 = factor(C3)
```

```
C4 = factor(C4)
C5 = factor(C5)
C6 = factor(C6)
C7 = factor(C7)
C8 = factor(C8)
C9 = factor(C9)
```

12.4 One-Way ANOVA and polynomial regression

```
asi <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab12-9.dat",
  sep="",col.names=c("x","y"))
attach(asi)
asi

# Polynomial model
xc = x -  mean(x)
xc2 = xc * xc
xc3 = xc2 * xc
xc4 = xc3 * xc
ft <- lm(y ~ xc + xc2 + xc3 + xc4)
ftp=summary(ft)
ftp
anova(ft)
```

The `summary(ft)` command gives the table of coefficients presented in the book. The `anova(ft)` command gives the sequential sums of squares *F* tests presented in Table 12.12 of the book. How to construct Table 12.11 should be self-evident by now and is of secondary importance anyway.

An alternative way to construct the sequential tests is by using R's capability for fitting orthogonal polynomials. We will not go into details but notice that the *P* values in the table of coefficients `summary(fto)` below agree with the earlier *P* values from `anova(ft)`.

```
fto <- lm(y ~ poly(x, degree = 4))
ftop=summary(fto)
ftop
anova(fto)
```

The square of the *t* values from `summary(fto)` should be the *F* values from `anova(ft)`.

All of the tests for the fourth degree term, whether from `summary(ft)`, `anova(ft)`, or `summary(fto)` should have the same *P* value. Except for the fourth degree term, the *P* values from `summary(ft)` are not expected to agree with those from `anova(ft)` and `summary(fto)`. In this example it is merely a coincidence that they nearly agree for the third degree term also.

12.4.1 Fisher's lack-of-fit test

See also Section 8.6.

```
# SLR
ft <- lm(y ~ x)
fitp=summary(ft)
ftp
anova(ft)

# One-way
```

```
G=factor(x)
fit <- lm(y ~ G-1)
fitp=summary(fit)
fitp
anova(ft,fit)
```

12.4.2 Figures

This code uses output from the previous subsection.

```
#Figure 12.6
plot(x,y)
#Figure 12.7
PL=c(4,6,8,10,12)
plot(PL,fit$coef)
#Figure 12.8 and 9
plot(x,rstandard(ft),xlab="Plate Length",ylab="Standardized residuals",
main="Residual-Plate Length plot")
qqnorm(rstandard(ft),ylab="Standardized residuals")
# Wilk-Francia Statistic
rankit=qnorm(ppoints(rstandard(ft),a=I(3/8)))
ys=sort(rstandard(ft))
Wprime=(cor(rankit,ys))**2
Wprime
#Figure 12.10, 11, and 12
plot(x,rstandard(fit),xlab="Plate Length",ylab="Standardized residuals",
main="Residual-Plate Length plot")
plot(fit$fit,rstandard(fit),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
qqnorm(rstandard(fit),ylab="Standardized residuals")
# Wilk-Francia Statistic
rankit=qnorm(ppoints(rstandard(fit),a=I(3/8)))
ys=sort(rstandard(fit))
Wprime=(cor(rankit,ys))**2
Wprime
```

12.5 Weighted least squares

This section needs checking

```
asi <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab12-9.dat",
  sep="",col.names=c("x","y"))
attach(asi)
asi

fit <- lm(y ~ x)
fitp=summary(fit)
fitp
ii=factor(x)
pe= lm(y ~ ii-1)
anova(lm(y ~ 1),fit,pe)

# Create the data used in WLS example
```

```
pex <- lm(x ~ ii-1)
ymeans <- pe$coef
xmeans <- pex$coef
wts <- xmeans + 7 - xmeans   # A column of 7s.

# Fit WLS model
wls <- lm(ymeans ~ xmeans, wt=wts)
summary(wls)
anova(lm(ymeans ~ 1),wls)
```

Compare the least squares estimates from the two `summary` commands, also the lack of fit error term with the weighted least squares error term.

12.5.1 *Unbalanced weights*

```
trucks <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab6-8.dat",
  sep="",col.names=c("x","y"))
attach(trucks)
trucks

fit <- lm(y ~ x)
fitp=summary(fit)
fitp
ii=factor(x)
pe= lm(y ~ ii-1)
anova(lm(y ~ 1),fit,pe)

# Enter the data used in WLS example
ymeans = c(172.5, 664.3, 633.0, 900.3, 1202.0, 987.0, 1068.5)
x = c(0.5, 1.0, 4.0, 4.5, 5.0, 5.5, 6.0)
wts = c(2, 3, 3, 3, 3, 1, 2)

# Fit WLS model
wls <- lm(ymeans ~ x, wt=wts)
summary(wls)
anova(lm(ymeans ~ 1),wls)
```

Compare the least squares estimates from the two `summary` commands, also the lack of fit error term with the weighted least squares error term.

12.5.2 *Nondiagonal case*

```
# At some point you need to have run   install.packages("MASS")
library(MASS)
lm.gls(formula, data, W, subset, na.action, inverse = FALSE,
method = "qr", model = FALSE, x = FALSE, y = FALSE,
contrasts = NULL, ...)

W=weight matrix, inverse=false
```

Chapter 13

Multiple Comparisons

Obviously, this has hardly been started.

Throughout the chapter we assume that the following commands have been run.

```
mand <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab12-4.dat",
sep="",col.names=c("yy","C2","C3","C4","C5","C6","C7","C8","C9"))
attach(mand)
mand

ii = factor(C2)
y = log(yy)
fit <- lm(y ~ ii)

fitp <- summary(fit)
fitp
anova(fit)

C3 = factor(C3)
C4 = factor(C4)
C5 = factor(C5)
C6 = factor(C6)
C7 = factor(C7)
C8 = factor(C8)
C9 = factor(C9)
```

Need to double check that this agrees with Section 12.4.

13.1 "Fisher's" LSD

Do the overall F test first. If and only if that is significant, do individual tests. For pairwise comparisons in a one-way ANOVA, you can use `pairwise.t.test(y, ii)`. Pairwise comparisons are not to be confused with the paired comparisons procedure of Section 4.1.

13.2 Bonferroni adjustments

Use appropriately adjusted quantiles. For pairwise comparisons in a one-way ANOVA, you can use `pairwise.t.test(y, ii, p.adj = "bonf")`.

13.3 Scheffé's method

No new commands, just some programming.

```
install.packages("agricolae")
```

```
library(agricolae)
scheffe.test(fit,"ii",group=TRUE)
MSE = deviance(fit)/df.residual(fit)
MSE
Fc <- anova(fit)["ii",4]
scheffe.test(y, ii, df.residual(fit), fit$sigma^2, Fc, alpha = 0.05, group=TRUE, mai
```

13.4 Studentized range methods

$$Q_{obs} = \frac{4.9031 - 4.0964}{\sqrt{0.00421/4}} =$$

The P value is

$$\Pr[Q > 24.87] = 0.000$$

The way you would compute the P values is by running `ptukey(24.87,7,39,lower.tail=FALSE)`.

13.4.1 Honest John's significant difference

To find the percentile of the studentized range distribution use in the book. $4.39 = Q(0.95, 7, 40) =$ `qtukey( 0.95, nmeans = 7, df = 40) = qtukey(0.95,7,40)`. However, what we really want is $Q(0.95, 7, 39) =$ `qtukey( 0.95, nmeans = 7, df = 39)`.

To fully automate the process, **one or more of these should work**.

```
TukeyHSD(aov(y ~ ii))
TukeyHSD(lm(y ~ ii))
TukeyHSD(fit)
TukeyHSD(summary(fit))
plot(TukeyHSD(aov(y ~ ii)))


# Balanced two-way, main effects only.
TukeyHSD(summary(y~ii+jj),"jj", "ii")
```

Chapter 14

Two-Way ANOVA

14.1 Unbalanced Two-way ANOVA

Code for the basic analysis including the diagnostic table and plots. The output name `fisher.acv` has no meaning here, it is borrowed from the next chapter where an analysis of covariance is performed on data presented by R.A. Fisher

```
rm(list = ls())
fisher.acv <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab14-1.dat",
   sep="",col.names=c("Litter","Mother","Weight"))
attach(fisher.acv)
fisher.acv
#summary(fisher.acv)
# Weight[12]=NA
# Weight[which(Litter == "I" & Mother == "A")]=NA

#Summary tables
#L=factor(Litter)     # Doing this wouldn't hurt but Litter and
#M=factor(Mother)     # Mother are nonnumeric, so this is not needed
cr <- lm(Weight ~ Litter:Mother)
crp=summary(cr)
crp
anova(cr)


infv = c(Weight,cr$fit,hatvalues(cr),rstandard(cr),rstudent(cr),cooks.distance(cr))
inf=matrix(infv,I(crp$df[1]+crp$df[2]),6,dimnames = list(NULL,
                              c("y", "yhat", "lev","r","t","C")))
inf



par(mfrow=c(2,2))
qqnorm(rstandard(cr),ylab="Standardized residuals")
plot(cr$fit,rstandard(cr),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
boxplot(rstandard(cr)~Litter,xlab="Litters",
ylab="Standardized residuals",main="Residual-Litters plot")
boxplot(rstandard(cr)~Mother,xlab="Mothers",
ylab="Standardized residuals",main="Residual-Mothers plot")
```

```
# Wilk-Francia
rankit=qnorm(ppoints(rstandard(cr),a=I(3/8)))
ys=sort(rstandard(cr))
Wprime=(cor(rankit,ys))**2
Wprime

i=seq(1,61,1)
Leverage=hatvalues(cr)
par(mfrow=c(2,2))
plot(i,Leverage,main="Leverage index plot")
plot(i,rstudent(cr),main="t residual index plot")
plot(i,cooks.distance(cr),main="Cook's distance index plot")


# This gives group means
cr <- lm(Weight ~ Litter:Mother-1)
crp=summary(cr)
crp

# These give Table 14.3
cr <- lm(Weight ~ Mother + Litter + Mother:Litter)
anova(cr)
cr <- lm(Weight ~ Litter + Mother + Litter:Mother)
anova(cr)

# These give Table 14.2
LM <- lm(Weight ~ Mother + Litter + Mother:Litter)
LMp=summary(LM)
LM.cp=((LMp$sigma**2 * LMp$df[2])/LMp$sigma**2)-LMp$df[2]+LMp$df[1]
LM.cp

L.M <- lm(Weight ~ Mother + Litter)
L.Mp=summary(L.M)
L.M.cp=((L.Mp$sigma**2 * L.Mp$df[2])/LMp$sigma**2)-L.Mp$df[2]+L.Mp$df[1]
L.M.cp

L <- lm(Weight ~ Litter)
Lp=summary(L)
L.cp=((Lp$sigma**2 * Lp$df[2])/LMp$sigma**2)-Lp$df[2]+Lp$df[1]
L.cp

M <- lm(Weight ~ Mother)
Mp=summary(M)
M.cp=((Mp$sigma**2 * Mp$df[2])/LMp$sigma**2)-Mp$df[2]+Mp$df[1]
M.cp

G <- lm(Weight ~ 1)
Gp=summary(G)
G.cp=((Gp$sigma**2 * Gp$df[2])/LMp$sigma**2)-Gp$df[2]+Gp$df[1]
G.cp

Cpinf = c("[LM]","[L][M]","[L]","[M]","[G]",
LMp$sigma**2 * LMp$df[2], L.Mp$sigma**2 * L.Mp$df[2], Lp$sigma**2 * Lp$df[2], Mp$sig
LMp$df[2], L.Mp$df[2], Lp$df[2], Mp$df[2], Gp$df[2],
LM.cp, L.M.cp, L.cp, M.cp, G.cp)
CpTable=matrix(Cpinf,5,4,dimnames = list(NULL,
                              c("Model","SSE","dfE","C_p")))
CpTable
```

The command `summary(cr)` produces the table of coefficients for the various fitted models. Note that side conditions are incorporated so that the software can produce unique estimates of the model coefficients. In fact, the side conditions used for the models `Litter:Mother` and `Litter + Mother + Litter:Mother` are not even consistent. The former sets the coefficient for Litter J and Mother J (`LitterJ:MotherJ`) equal to 0, making the the intercept estimate the mean for Litter J and Mother J and all of the other coefficients are their group mean minus the Litter J, Mother J mean. The latter sets every coefficient involving group A equal to 0, making the intercept estimate the mean for Litter A and Mother A. The other coefficients have, for example, the mean for Litter I, Mother J being the sum of the estimated effects for intercept, `LitterI`, `MotherJ` and `LitterI:MotherJ`.

The P values for pairwise comparisons in the one way ANOVA on Mothers can be obtained using

```
#pairwise.t.test(Weight,Mother,p.adjust.method="bonferroni")
pairwise.t.test(Weight,Mother,p.adjust="none")
```

To perform the analysis with case 12 deleted, AFTER READING IN THE DATA run the command `Weight[12]=NA` *and then repeat the code given earlier.* To drop out the entire Litter I, Mother A group you can use `Weight[which(Litter == "I" & Mother == "A")]=NA`.

14.2 Modeling contrasts

Because of how R's `lm` command chooses side conditions for reporting the table of coefficients, the first three entries in the unnumbered table of pairwise comparisons (comparing the other Mother groups to Mother A) are available from fitting

```
cr1 <- lm(Weight ~ Litter + Mother)
summary(cr1)
```

The Bonferroni P values reported in the book are the number of pairwise comparisons, i.e. $\binom{4}{2} = 6$, times the P values reported here, with values exceeding 1 rounded off to 1. Finding the last three Mother pairwise comparisons requires the regression fitting of the next section.

14.2.1 Nonequivalence of Tests

We now construct (with the help of Fletcher Christensen) the factor variables needed to obtain the F statistics in the displays of Subsection 14.2.1.

```
# Construct the factor variable for mothers that does
# not distinguish between groups A and F
MomAF = Mother
MomAF[which(MomAF == "F")]="A"
MomAF

# Construct the factor variable for mothers that does
# not distinguish between groups I and J
MomIJ = Mother
MomIJ[which(MomIJ == "I")]="J"
```

```
MomIJ

# Construct the factor variable for mothers that does
# not distinguish between groups A and F or between groups I and J
MomAF.IJ = MomAF
MomAF.IJ[which(MomAF.IJ == "I")]="J"
MomAF.IJ

# Not used in book but
# construct a factor variable for mothers that does
# not distinguish between groups A, F, and J
MomAFJ = Mother
MomAFJ[which(MomAFJ == "F" | MomAFJ == "J")]="A"
MomAFJ
```

Given these new factor variables we obtain the *displayed F* tests in Subsection 14.2.1. Recall that these tests illustrate that *F* tests for whether Mothers A and F have the same effect change depending on the other assumptions built into the model. In the `anova( , , )` commands, our interest is in the penultimate *F* test.

```
# Subsection 14.2.1, first display
cr <- lm(Weight ~ Mother:Litter)
cr1 <- lm(Weight ~ Mother)
# These give Table 14.3
cr2 <- lm(Weight ~ MomAF)
anova(cr2,cr1,cr)

# Second and third displays
cr <- lm(Weight ~ Mother:Litter)
cr1 <- lm(Weight ~ Mother + Litter)
cr2 <- lm(Weight ~ MomAF+Litter)
anova(cr2,cr1,cr)
anova(cr2,cr1)

# Last display
cr <- lm(Weight ~ Mother:Litter)
cr1 <- lm(Weight ~ MomIJ + Litter)
cr2 <- lm(Weight ~ MomAF.IJ+Litter)
anova(cr2,cr1,cr)
```

14.3 Regression modeling

We need to create the indicator variables X1 to X8. Once we have the indicator variables, the rest of the computing in this section is straightforward.

One way create the indicator variables is to create 0 vectors of the correct length and then identify which terms in the vector should be 1. Having previously read in the data.

```
J=Mother-Mother+1
X1=J-J
X2=X1
X3=X1
X4=X1
X5=X1
X6=X1
```

```
X7=X1
X8=X1
X1[which(Litter == "A")]=1
X2[which(Litter == "F")]=1
X3[which(Litter == "I")]=1
X4[which(Litter == "J")]=1

X5[which(Mother == "A")]=1
X6[which(Mother == "F")]=1
X7[which(Mother == "I")]=1
X7[which(Mother == "J")]=1

# turn factor variables X1 and X5 into numeric variables.
X1=as.numeric(X1)
X2=as.numeric(X2)
X3=as.numeric(X3)
X4=as.numeric(X4)
X5=as.numeric(X5)
X6=as.numeric(X6)
X7=as.numeric(X7)
X8=as.numeric(X8)
```

Alternatively, we could use the model matrix from fitting the model but unfortunately the model matrix has been adjusted for the side conditions that R chooses to use, so we will still need to define X1 and X5 directly.

```
rm(list = ls())
fisher.acv <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab14-1.dat",
   sep="",col.names=c("Litter","Mother","Weight"))
attach(fisher.acv)
fisher.acv
#summary(fisher.acv)

#Summary tables
#L=factor(Litter)     # Doing this wouldn't hurt but Litter and
#M=factor(Mother)     # Mother are nonnumeric, so this is not needed
cr <- lm(Weight ~ Litter+Mother)
crp=summary(cr)
crp
anova(cr)

# Define indicator variables for the subsection
X <- model.matrix(cr)
X
J=X[,1]
X2=X[,2]
X3=X[,3]
X4=X[,4]
X6=X[,5]
X7=X[,6]
X8=X[,7]
X1=J-J
X5=J-J
```

```
X1[which(Litter == "A")]=1
X5[which(Mother == "A")]=1
# turn factor variables X1 and X5 into numeric variables.
X1=as.numeric(X1)
X5=as.numeric(X5)

# Fit additive model using indicators
cr1=lm(Weight ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8)
summary(cr1)
anova(c1)


Z1=model.matrix(~Litter-1)
Z2=model.matrix(~Mother-1)
X1=Z1[,1]
X2=Z1[,2]
X2=Z1[,3]
X4=Z1[,4]
X5=Z2[,1]
X6=Z2[,2]
X7=Z2[,3]
X8=Z2[,4]
J=X1-X1+1
```

With X1 to X8 treated as regression variables, virtually all programs for fitting such a model will drop out variables X4 and X8 because they create linear dependencies with the previous variables. X4 and X8 are, respectively, the indicator variables for Litter J and Mother J, so dropping them make the other estimated effects into comparisons with those groups. Thus the rows of the Table of Coefficients reported for X5, X6, X7 agree with the entries for $-(\eta_J - \eta_A)$, $-(\eta_J - \eta_F)$, and $-(\eta_J - \eta_I)$ in the unnumbered table of pairwise comparisons from early in Section 14.2. We earlier obtained the entries for $\eta_F - \eta_A$, $\eta_I - \eta_A$, and $\eta_J - \eta_A$ by fitting an additive effects ANOVA model in R. The remaining effect that we need for this unnumbered table is $\eta_I - \eta_F$ which can be obtained dropping X6 (the indicator variable associated with Mother F). Fit

```
cr1=lm(Weight ~ X1 + X2 + X3 + X4 + X5 + X7  + X8)
summary(cr1)
```

and look at the row for X7 (Mother I). *As indicated earlier, to get the Bonferroni P values reported in the book, the P values reported here have to be multiplied by 6 (with truncation of larger numbers down to 1).*

14.4 Homologous factors

14.4.1 Symmetric additive effects

Again the computing is straightforward except that this section is using different names for the indicator variables defined in the previous section

```
rm(list = ls())
fisher.acv <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab14-1.dat",
   sep="",col.names=c("Litter","Mother","Weight"))
attach(fisher.acv)
fisher.acv
#summary(fisher.acv)
```

```
#Summary tables
#L=factor(Litter)    # Doing this wouldn't hurt but Litter and
#M=factor(Mother)    # Mother are nonnumeric, so this is not needed
cr <- lm(Weight ~ Litter+Mother)
crp=summary(cr)
crp
anova(cr)


# Repeat indicator variables as used in previous subsection
X <- model.matrix(cr)
X
J=X[,1]
X2=X[,2]
X3=X[,3]
X4=X[,4]
X6=X[,5]
X7=X[,6]
X8=X[,7]
X1=J-J
X5=J-J
X1[which(Litter == "A")]=1
X5[which(Mother == "A")]=1
# turn factor variables X1 and X5 into numeric variables.
X1=as.numeric(X1)
X5=as.numeric(X5)

# Make indicator variable notation agree with this subsection
La = X1
Lf = X2
Li = X3
Lj = X4
Ma = X5
Mf = X6
Mi = X7
Mj = X8
# Define new variables used in subsection.
A=La + Ma
F=Lf + Mf
I=Li + Mi
J=Lj + Mj

# Obtain output for models fitted in subsection
cr1=lm(Weight ~ La + Lf + Li + Lj + Ma + Mf + Mi + Mj)
anova(cr1)
cr1=lm(Weight ~ La + Lf + Li + Ma + Mf + Mi)
anova(cr1)
cr2=lm(Weight ~ A + F + I + J)
anova(cr2)
cr2=lm(Weight ~ A + F + I)
anova(cr2)
anova(cr2,cr1)
```

```
cr <- lm(Weight ~ Litter:Mother)
anova(cr2,cr1,cr)
```

14.4.2 Skew symmetric additive effects

Add the following to the code of the previous subsection.

```
ssa=(La - Ma)
ssf=(Lf - Mf)
ssi=(Li - Mi)
ssj=(Lj - Mj)
cr3=lm(Weight ~ ssa + ssf + ssi + ssj)
summary(cr3)
anova(cr3)
anova(cr3,cr1)
```

14.4.3 Symmetry

First we change the four Litter and Mother categories identified by letters into numbers: 1 to 4 for `ii` and 0 to 3 for`jj`. Then we change the pairs of numbers into a single index $r = 1, \dots, 16$. Finally, as per Table 14.6, we modify r into the factor variable s that treats Litters and Mothers symmetrically. The remainder of the computing is straightforward.

```
ii=Litter
jj=Mother
ii[which(Litter == "A")]=1
ii[which(Litter == "F")]=2
ii[which(Litter == "I")]=3
ii[which(Litter == "J")]=4
jj[which(Mother == "A")]=0
jj[which(Mother == "F")]=1
jj[which(Mother == "I")]=2
jj[which(Mother == "J")]=3
# Next turn factor variables ii and jj into numeric index variable r.
ii=as.numeric(ii)
jj=as.numeric(jj)
r=ii+(4*jj)
# Construct symmetry index s.
s=r
s[which(r == 5)]=2
s[which(r == 9)]=3
s[which(r == 10)]=7
s[which(r == 13)]=4
s[which(r == 14)]=8
s[which(r == 15)]=12

# Turn numeric variables r and s into factor variables.
r=factor(r)
s=factor(s)
cr=lm(Weight ~ r)
anova(cr)
cr1=lm(Weight ~ s)
anova(cr1,cr)
```

Chapter 15

ACOVA and Interactions

15.1 One covariate example

```
rm(list = ls())
fisher.acv <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab15-1.dat",
sep="",col.names=c("Body","Heart","sex"))
attach(fisher.acv)
fisher.acv
#summary(fisher.acv)

#Summary tables
Sex=factor(sex)
cr <- lm(Heart ~ Sex+Body)
crp=summary(cr)
crp
anova(cr)

confint(cr)
# or compute confidence intervals yourself
R=crp$cov.unscaled
se <- sqrt(diag(R)) * crp$sigma
ci = c(cr$coef - qt(.975,crp$df[2])*se,cr$coef + qt(.975,crp$df[2])*se)
CI95 = matrix(ci,crp$df[1],2)
CI95
# This code works for ``lm'' but not ``aov''


#prediction Both sexes, Body=2.6
new = data.frame(Sex=c("1","2"),Body=c(2.6,2.6))
predict(cr,new,se.fit=T,interval="confidence")
predict(cr,new,interval="prediction")







infv = c(y,cr$fit,hatvalues(cr),rstandard(cr),rstudent(cr),
     cooks.distance(cr))
inf=matrix(infv,I(crp$df[1]+crp$df[2]),6,dimnames = list(NULL,
  c("y", "yhat", "lev","r","t","C")))
inf
```

```
# delete y from table if missing observations

qqnorm(rstandard(cr),ylab="Standardized residuals")

# Wilk-Francia
rankit=qnorm(ppoints(rstandard(cr),a=I(3/8)))
ys=sort(rstandard(cr))
Wprime=(cor(rankit,ys))**2
Wprime


plot(cr$fit,rstandard(cr),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
plot(sex,rstandard(cr),xlab="x",
ylab="Standardized residuals",main="Residual-Sex plot")
```

15.2 Regression modeling

15.3 ACOVA and two-way ANOVA

```
rm(list = ls())
fisher.acv <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab15-6.dat",
sep="",col.names=c("Times","Days","Wt"))
attach(fisher.acv)
fisher.acv
#summary(fisher.acv)

#Summary tables
Times2=Times**2
Day=factor(Days)
Time=factor(Times)
#cr <- lm(Wt ~ Day)
cr <- lm(Wt ~ Day + Times + Times2 )
#cr <- lm(Wt ~ Day + Time )
#cr <- lm(Wt ~ Day + Times )
crp=summary(cr)
crp
anova(cr)
```

15.4 Near replicate lack-or-fit tests

This code is about clustering cases to identify near replicates. It uses some standard R and also the package `cluster`.

https://cran.r-project.org/web/packages/cluster/index.html

```
install.packages("cluster")
library(cluster)
coleman <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab6-4.dat",
sep="",col.names=c("School","x1","x2","x3","x4","x5","y"))
attach(coleman)
```

```
coleman
X=coleman[,-1]
X=X[,-6]
X
nr=agnes(X,diss=F,method="average")
# other methods:  single, complete, ward
plot(nr,which.plots=2)
nrep=cutree(nr,k=12)  #cutree(as.hclust(nr), h = 6))
nrep
nr2=pam(X,12)
nr2$clustering
nr3=kmeans(X,12)
nr3$cluster

nearrep=lm(y ~ x1+x2+x3+x4+x5 + nrep)
summary(nearrep)
nearrep2=lm(y ~ x1+x2+x3+x4+x5 + nr2$clustering)
summary(nearrep2)
nearrep3=lm(y ~ x1+x2+x3+x4+x5 + nr3$cluster)
summary(nearrep3)
```

running `kmeans` repeatedly does not always give the same answer. `pam` is a robust version of `kmeans`.

15.5 Exercises

```
rm(list = ls())
fisher.acv <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab15-9.dat",
sep="",col.names=c("Temperature","Trees","y","z"))
attach(fisher.acv)
fisher.acv
#summary(fisher.acv)

#Summary tables
Temp=factor(Temperature)
Tree=factor(Trees)
cr <- lm(y ~ z + Temp + Tree)
crp=summary(cr)
crp
anova(cr)
```

Chapter 16

Multifactor Structures

16.1 Unbalanced Three-way: Moisture data

The following code shows how to fit a variety of the models used in this section. At the end, for the full interaction model it also includes output from `stepAIC`. The stepAIC output drops the three-factor interaction, then it checks to drop a two-factor interaction, then with only one two-factor in the model it checks to see if it can drop that two-factor or the main effect not in the two-factor at which point it stops.

```
rm(list = ls())
scheffe <- read.table(
"C:\\E-drive\\Books\\ANREG2\\newdata\\tab16-1.dat",
sep="",col.names=c("y","a","b","c"))
attach(scheffe)
scheffe
summary(scheffe)

A=factor(a)
B=factor(b)
C=factor(c)
# The role of "x" in the text is played by "b" here.
b2=b*b

# Model [ABC]
sabc <- lm(y ~ A:B:C)
coef=summary(sabc)
coef
anova(sabc)

# Full factorial version of [ABC]
sabc <- lm(y ~ A*B*C)
#coef=summary(sabc)
#coef
anova(sabc)

# No code for the other models
# in Table 16.2 other than

# Model [AB][C]
sabc <- lm(y ~ A:B + C-1)
#coef=summary(sabc)
#coef
```

```
anova(sabc)

# Model [A_0][A_1][A_2][C] (equivalent to above)
# b plays the role played by x in the book
sabc <- lm(y~A + A:b + A:b2 + C -1)
#coef=summary(sabc)
#coef
anova(sabc)

# [A_0][A_1][A_2][C] Alternative syntax
sabc <- lm(y~A/(b+I(b^2))+C -1)
#coef=summary(sabc)
#coef
anova(sabc)

# [A_0][A_1][C]
sabc <- lm(y ~ A + A:b + C - 1)
coef=summary(sabc)
coef
anova(sabc)

# [A_0][A_1][C] alternative syntax
sabc <- lm(y ~ A/b + C - 1)
coef=summary(sabc)
coef
anova(sabc)

# Generate the variable A2 discussed in text's Table 16.7
# and in next Subsection.
# Do not confuse variable A2 with model effect [A_2].
# They accomplish very different things.
A2=A
A2[(A2 == 3)]  <- 2

# stepwise output
sabc <- lm(y ~ A*B*C)
library(MASS)
stepAIC(sabc)
```

The fact that the slopes in $[A_0][A_1][C]$ are not all the same for the different kinds of salt constitutes [AB] interaction. See the next subsection for how to fit the more unusual models involving variable A2 discussed in the book.

In retrospect, A2 was a bad choice of name for the variable that does not distinguish between levels 2 and 3 of the salts (factor A) because of its potential for confusion with the model effect $[A_2]$ which indicates inclusion in our models of different quadratic (hence the 2 in $[A_2]$) effects for different salts (hence the A in $[A_2]$).

16.1.1 Computing

Because it is the easiest program I know, most of the analyses in this book were done in Minitab. We now present and contrast R and SAS code for fitting $[AB][C]$ and discuss the fitting of other models from this section. Table 16.7 illustrates the variables needed for a full analysis. The online data file contains only the y values and indices for the three groups. Creating X and X2 is generally easy. (In

Table 16.1: *Moisture data, indices, and predictors.*

	A	B	C	X	X2	A2		A	B	C	X	X2	A2
y	i	j	k	x	x^2		y	i	j	k	x	x^2	
8	1	1	1	1	1	1	11	1	2	2	2	4	1
17	1	2	1	2	4	1	16	1	3	2	3	9	1
22	1	3	1	3	9	1	3	2	1	2	1	1	2
7	2	1	1	1	1	2	17	2	2	2	2	4	2
26	2	2	1	2	4	2	32	2	3	2	3	9	2
34	2	3	1	3	9	2	5	3	1	2	1	1	2
10	3	1	1	1	1	2	16	3	2	2	2	4	2
24	3	2	1	2	4	2	33	3	3	2	3	9	2
39	3	3	1	3	9	2	4	1	1	2	1	1	1
13	1	2	1	2	4	1	10	1	2	2	2	4	1
20	1	3	1	3	9	1	15	1	3	2	3	9	1
10	2	1	1	1	1	2	5	2	1	2	1	1	2
24	2	2	1	2	4	2	19	2	2	2	2	4	2
9	3	1	1	1	1	2	29	2	3	2	3	9	2
36	3	3	1	3	9	2	4	3	1	2	1	1	2
5	1	1	2	1	1	1	34	3	3	2	3	9	2

the code given earlier b=X and b2=X2.) Creating the variable A2 that does not distinguish between salts 2 and 3 can be trickier. If we had a huge number of observations, we would want to write a program to modify A into A2. With the data we have, in Minitab it is easy to make a copy of A and modify it appropriately in the spreadsheet. Similarly, it is easy to create A2 in R using `A2=A` followed by `A2[(A2 == 3)] <- 2`. For SAS, I would probably modify the data file so that I could read A2 with the rest of the data.

To fit the other models, one needs to modify the part of the code that specifies the model. In R this involves changes to "`sabc <- lm(y ~ A:B + C)`" and in SAS it involves changes to "`model y = A*B C;`". Alternative model specifications follow.

Model	Minitab	R	SAS
$[ABC]$	$A\vert B\vert C$	A:B:C-1	A*B*C
$[AB][BC]$	$A\vert B \quad B\vert C$	A:B+B:C-1	A*B B*C
$[AB][C]$	$A\vert B \quad C$	A:B+C-1	A*B C
$[A_0][A_1][A_2][C]$	$A\vert X \quad A\vert X2 \quad C$	A+A:X+A:X2+C	A A*X A*X2 C
$[A_0][A_1][C]$	$A \quad A\vert X \quad C$	A+A:X+C-1	A A*X C/noint
$[A_0][A_1][C], A_{21}=A_{31}$	$A \quad A2\vert X \quad C$	A+A2:X+C-1	A A2*X C/noint
$[A_0][A_1][C], A_{21}=A_{31}, A_{20}=A_{30}$	$A2 \quad A2\vert X \quad C$	A2+A2:X+C-1	A2 A2*X C

For our earlier R code, the model $[A_0][A_1][C]$, $A_{21}=A_{31}$, $A_{20}=A_{30}$ would be written `A2+A2:b+C-1` rather than as above because `b` and `b2` play the roles of X and X2. Also fitting the model term $[A_0]$ without the restriction that $A_{20}=A_{30}$ is equivalent to using the factor variable `A` in R.

SAS code for fitting $[AB][C]$ follows. The code assumes that the data file is the same directory (folder) as the SAS file.

```
options ps=60 ls=72 nodate;
data anova;
   infile 'tab16-1.dat';
   input y A B C;
   X = B;
   X2=X*X;
proc glm data=anova;
   class A B C ;
   model y = A*B C ;
   means C / lsd alpha=.01 ;
   output out=new r=ehat p=yhat cookd=c h=hi rstudent=tresid student=sr;
```

```
proc plot;
   plot ehat*yhat sr*R/  vpos=16 hpos=32;
proc rank data=new normal=blom;
   var sr;
   ranks nscores;
proc plot;
   plot sr*nscores/vpos=16 hpos=32;
run;
```

16.2 Balanced three-way: Abrasion resistance data

```
rm(list = ls())
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab16-8.dat",
sep="",col.names=c("y","s","f","p","rep"))
attach(abraid)
abraid
summary(abraid)

S=factor(s)
F=factor(f)
P=factor(p)
p2=p*p
bsfp <- lm(y ~ S:F:P)
coef=summary(bsfp)
coef
anova(bsfp)

bsfp <- lm(y ~ (S+F+P)^3)
coef=summary(bsfp)
coef
anova(bsfp)

qqnorm(rstudent(bsfp),ylab="Standardized residuals")
plot(bsfp$fit,rstudent(bsfp),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")




# Wilk-Francia
rankit=qnorm(ppoints(rstandard(bsfp),a=I(3/8)))
ys=sort(rstandard(bsfp))
Wprime=(cor(rankit,ys))**2
Wprime


# conf. int. for cell 1,1,1
new = data.frame(S=c("1"),F=c("1"),P=c("1"))
predict(bsfp,new,se.fit=T,interval="confidence",level=0.95)


bsfp <- lm(y ~ S:F + F:P   -1)
bsfp <- lm(y ~ S:F + F:p + F:p2 -1)
bsfp <- lm(y ~ S:F + F:p  -1)

#Summary tables
S=factor(s)
F=factor(f)
P=factor(p)
p2=p*p
p0=c(1, 2, 3, 1, 2, 3, 0, 0, 0, 0, 0, 0, 1, 2, 3, 1, 2, 3, 0, 0, 0, 0, 0, 0)
bsfp <- lm(y ~ S:F + p0  -1)

sf= c(11, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12, 21, 21, 21, 21, 21, 21,
  21, 21, 21, 21, 21, 21)
SF=factor(sf)
bsfp <- lm(y ~ SF + p0 +  -1)
coef=summary(bsfp)
coef
anova(bsfp)
```

The book contains plots of the proportion polynomials fitted here. In general with just the [SF] and FP] interactions, for each level of Fill (here A and B but there could be more) you will get parallel curves (here parabolas) for each level of S. There is no interaction between S and P when you look only at a fixed level of F. When you change the level of F (go from Fill A to Fill B) you again get parallel curves BUT the shapes of the Fill B curves can be different from those for Fill A AND the distances between different curves can be different from what they were with Fill A.

Distances between the curves are the S effects but they can (have to) change when you change the Fill. That is [SF] interaction. The parabolas model the P effects and their shapes have to change depending on the Fill. That is the [FP] interaction.

An [SFP] interaction means each one of the 12 groups can do whatever it wants. One interesting way to examine that would be to fit a separate parabola in proportions for every one of the 4 SF combinations. (3 regression parameters per parabola times 4 different parabolas give 12 different parameters to explain the 12 groups.)

You could plot these 4 parabolas in a figure like 16.6 and see if you can identify any similarities among them.

16.3 Higher order structures

It would be interesting to see how well `stepAIC` works on higher-order structures.

Chapter 17

Basic Experimental Designs

17.4 Randomized complete block designs

```
rm(list = ls())
lenth.rcb <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab17-1.dat",
sep="",col.names=c("b","t","y"))
attach(lenth.rcb)
lenth.rcb
#summary(lenth.rcb)

#Summary tables
B=factor(b)
T=factor(t)
cr <- lm(y ~ B+T)
crp=summary(cr)
crp
anova(cr)

#compute confidence intervals
R=crp$cov.unscaled
se <- sqrt(diag(R)) * crp$sigma
ci = c(cr$coef - qt(.975,crp$df[2])*se,cr$coef + qt(.975,crp$df[2])*se)
CI95 = matrix(ci,crp$df[1],2)
CI95



#prediction
new = data.frame(B=c("1","1","1","1"),T=c("1","2","3","4"))
predict(cr,new,#se.fit=T,
           interval="confidence")
predict(cr,new,interval="prediction")



infv = c(y,cr$fit,hatvalues(cr),rstandard(cr),rstudent(cr),
     cooks.distance(cr))
inf=matrix(infv,I(crp$df[1]+crp$df[2]),6,dimnames = list(NULL,
  c("y", "yhat", "lev","r","t","C")))
inf
# delete y from table if missing observations
```

```
infv = c(y,cr$fit,b,t,hatvalues(cr),rstandard(cr),rstudent(cr),
     cooks.distance(cr))
inf=matrix(infv,I(crp$df[1]+crp$df[2]),8,dimnames = list(NULL,
  c("y", "yhat", "Blks","Trts","lev","r","t","C")))
inf
# delete y from table if missing observations

qqnorm(rstandard(cr),ylab="Standardized residuals")
plot(cr$fit,rstandard(cr),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")

plot(b,rstandard(cr),xlab="Blocks",
ylab="Standardized residuals",main="Residual-Block plot")



Leverage=hatvalues(cr)
plot(10*b+t,Leverage,main="Leverage plot")

# Wilk-Francia
rankit=qnorm(ppoints(rstandard(cr),a=I(3/8)))
ys=sort(rstandard(cr))
Wprime=(cor(rankit,ys))**2
Wprime
```

17.4.1 Paired comparisons

In Section 17.4.1 of the book we mention that the paired comparison procedure is equivalent to fitting a randomized complete block model. To do this we need not only the (x,y) data structure but a third variable, say, z that identifies the pair. x and z need to be factor variables. Fit the two-way ANOVA

```
dr <- lm(y ~ x + z)
drp=summary(dr)
drp
anova(dr)
```

and look at the tests associated with z to establish differences between samples. We do not have a data file set up for this analysis.

17.5 Latin squares

```
root <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab17-4.dat",
 sep="",col.names=c("y","Rows","Cols","Trts"))
attach(root)
T=factor(Trts)
R=factor(Rows)
C=factor(Cols)
fit <- lm(y ~ C + R + T)
anova(fit)

T
Contrast = gsub("[^4]","1",T)     #Change anything that is not a 4 into a 1
Contrast = factor(Contrast)
Contrast                     #Check that this did what it was supposed to
fit <- lm(y ~ C + R + Contrast)
anova(fit)
```

17.6 Balanced incomplete blocks

```
rm(list = ls())
dish <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab17-7.dat",
   sep="",col.names=c("Blk","Trt","y"))
attach(dish)
Blocks = factor(Blk)
Trt = factor(Trt)
fit <- lm(y ~ Blocks + Trt)
fitp <- summary(fit)
anova(fit)

infv = c(y,fit$fit,hatvalues(fit),rstandard(fit),rstudent(fit),
     cooks.distance(fit))
inf=matrix(infv,I(fitp$df[1]+fitp$df[2]),6,dimnames = list(NULL,
  c("y", "yhat", "lev","r","t","C")))
inf
# delete y from table if missing observations
```

17.7 Youden squares

```
rm(list = ls())
root <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab17-11.dat",
  sep="",col.names=c("y","Rows","Cols","Trts"))
attach(root)
T=factor(Trts)
R=factor(Rows)
C=factor(Cols)
fit <- lm(y ~ R + C + T)
fitp <- summary(fit)
anova(fit)
infv = c(y,fit$fit,hatvalues(fit),rstandard(fit),rstudent(fit),
     cooks.distance(fit))
inf=matrix(infv,I(fitp$df[1]+fitp$df[2]),6,dimnames = list(NULL,
  c("y", "yhat", "lev","r","t","C")))
inf
# delete y from table if missing observations
```

Chapter 18

Factorial Treatments

18.1

18.2 RCB Analysis

We already had the chance to check assumptions on this model in the previous chapter.

```
lenth.rcb <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab18-1.dat",
sep="",col.names=c("b","t","y","Disk","Window"))
attach(lenth.rcb)
lenth.rcb

B=factor(b)
T=factor(t)
W=factor(Window)
D=factor(Disk)
fit <- lm(y ~ B+D*W)
fitp=summary(fit)
fitp
anova(fit)
fit <- lm(y ~ B+W*D)
anova(fit)
```

18.4 Interaction in a Latin square

```
rm(list = ls())
potato <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab18-2.dat",
sep="",col.names=c("Rows","Cols","Nit","Pot","y"))
attach(potato)
potato
# Create a factor variable for treatments since not in data file
Trts=10*Nit+Pot
Trts=c(4,1,0,5,3,2,1,2,4,3,0,5,5,3,2,4,1,0,0,4,1,2,5,3,2,5,3,0,4,1,3,0,5,1,2,4)

T=factor(Trts)
R=factor(Rows)
C=factor(Cols)
P=factor(Pot)
p=Pot
p2=p*p
N=factor(Nit)
```

```
p2=p*p

bsfp <- lm(y ~ R + C + T)
anova(bsfp)

par(mfrow=c(2,2))
plot(bsfp$fit,rstudent(bsfp),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
plot(Rows,rstudent(bsfp),xlab="Rows",
ylab="Standardized residuals",main="Residual-Rows plot")
plot(Cols,rstudent(bsfp),xlab="Columns",
ylab="Standardized residuals",main="Residual-Columns plot")
plot(Trts,rstudent(bsfp),xlab="Treatments",
ylab="Standardized residuals",main="Residual-Treatments plot")

par(mfrow=c(1,1))
qqnorm(rstudent(bsfp),ylab="Standardized residuals")
# Wilk-Francia
rankit=qnorm(ppoints(rstandard(bsfp),a=I(3/8)))
ys=sort(rstandard(bsfp))
Wprime=(cor(rankit,ys))**2
Wprime

bsfp <- lm(y ~ R + C + N*P)
anova(bsfp)
bsfp <- lm(y ~ R + C + N*p + N*p2)
anova(bsfp)
bsfp <- lm(y ~ R + C + N*p)
anova(bsfp)
coef=summary(bsfp)
coef
bsfp <- lm(y ~ R + C + N*p-p)
anova(bsfp)
coef=summary(bsfp)
coef
```

The last two models fitted are equivalent but have slightly different table of coefficients. The last one presents the slopes for each level of nitrogen directly. The penultimate model presents the slope for the low (first listed) level of nitrogen and the difference between the two slopes. The table of coefficients in the book is from Minitab and it presents the average slope and the number that you have to **add** to the average slope to get the slope for the low level of nitrogen. By subtracting this same number you get the slope for the high level of nitrogen. Note that the t test for the number you have to add in Minitab is the same as the t test for the difference between slopes in the penultimate model.

18.5 A balanced incomplete block design

```
rm(list = ls())
dish <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab18-5A.dat",
  sep="",col.names=c("Row","Blk","Trt","y","Det","Amt","Cntl"))
attach(dish)
Treatments = factor(Trt)
```

```
Blocks = factor(Blk)
Det = factor(Det)
Amount = factor(Amt)
Control = factor(Cntl)
a = Amt
a2=a*a
a3=a2*a
fit <- lm(y ~ Blocks + Treatments)
anova(fit)
fit <-  lm(y ~ Blocks + Control + Det*Amount)
anova(fit)
fit <-  lm(y ~ Blocks + Control + Det*a + Det*a2 + Det*a3)
anova(fit)

Det
# Det 3 is the control, 1=I,2=II
d1 = gsub("[^1]","0",Det)
d2 = gsub("[^2]","0",Det)
d3 = gsub("[^3]","0",Det)
d1 = as.numeric(d1)
d2 = as.numeric(d2)/2
d3 = as.numeric(d3)/3
matrix(c(d1,d2,d3),ncol=3)

fit <-  lm(y ~ Blocks + d3 + d1 + d1:a + d2:a + d1:a2 + d2:a2 + d1:a3 + d2:a3)
anova(fit)

fit <-  lm(y ~ Blocks + d3 + d1 + d1:a + d2:a + d1:a2)

fitp=summary(fit)
fitp
```

18.6 Extensions of Latin Squares

```
rm(list = ls())
milk <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab18-8.dat",
  sep="",col.names=c("y","Sq","Period","Col","Cow","Trt"))
attach(milk)
s=factor(Sq)
p=factor(Period)
c=factor(Col)
cw=factor(Cow)
t=factor(Trt)
fit <-  lm(y~s + s:c + s:p  + t)
fit <-  lm(y~s + s:c + p + s:p  + t)
fit <-  lm(y~cw + p   + t)
anova(fit)
```

Chapter 19

Dependent Data

19.1 Split plots

In longitudinal data analysis these models are known as random intercept models because every individual has a separate random effect associated with it.

19.1.1 Whole plot analysis

```
rm(list = ls())
garnw <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-1w.dat",
sep="",col.names=c("rep","l","y"))
attach(garnw)
garnw
summary(garnw)

Rep=factor(rep)
L=factor(l)

wp <- lm(y ~ Rep + L)
anova(wp)


par(mfrow=c(2,2))
qqnorm(rstudent(wp),ylab="Standardized residuals")
plot(wp$fit,rstudent(wp),xlab="Fitted",
ylab="Standardized residuals",main="Whole-Plot Residual-Fitted plot")
plot(rep,rstudent(wp),xlab="Replications",
ylab="Standardized residuals",main="Whole-Plot Residual-Replication plot")
plot(l,rstudent(wp),xlab="Laundries",
ylab="Standardized residuals",main="Whole-Plot Residual-Laundry plot")
```

The following uses `lm` to get all the pieces of the full ANOVA table. Error 2 is the reported Error. Error 1 is the effect `Rep:L`. The output uses Error 2 everywhere, even in places where we need it to use Error 1. *You need to compute the whole plot F statistics yourself.*

```
rm(list = ls())
garn <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-1.dat",
sep="",col.names=c("rep","l","t","y"))
attach(garn)
garn
summary(garn)
```

```
Rep=factor(rep)
L=factor(l)
T=factor(t)

ab <- lm(y ~ (Rep + L)^2 + T + T:L)
anova(ab)

abp <- summary(ab)
abp

infv = c(rep,l,t,y,ab$fit,hatvalues(ab),rstandard(ab),rstudent(ab),
     cooks.distance(ab))
inf=matrix(infv,I(abp$df[1]+abp$df[2]),9,dimnames = list(NULL,
  c("rep","l","t","y", "yhat", "lev","r","t","C")))
inf

#Residual Plots


par(mfrow=c(1,1))
qqnorm(rstudent(ab),ylab="Standardized residuals")


# Wilk-Francia
rankit=qnorm(ppoints(rstandard(ab),a=I(3/8)))
ys=sort(rstandard(ab))
Wprime=(cor(rankit,ys))**2
Wprime


par(mfrow=c(2,2))
plot(ab$fit,rstudent(ab),xlab="Fitted",
ylab="Standardized residuals",main="Subplot Residual-Fitted plot")
plot(rep,rstudent(ab),xlab="Replications",
ylab="Standardized residuals",main="Subplot Residual-Replication plot")
plot(l,rstudent(ab),xlab="Laundries",
ylab="Standardized residuals",main="Subplot Residual-Laundry plot")
plot(t,rstudent(ab),xlab="Tests",
ylab="Standardized residuals",main="Subplot Residual-Test plot")
```

19.1.2 Interaction plots

There is an “interaction.plot” command but it only works for balanced designs.

```
interaction.plot(Tests,Laundries,Fits)}


#Create a matrix with columns 1,2,3,5 from Table 19.3
inter = c(rep,t,l,ab$fit)
int=matrix(inter,I(abp$df[1]+abp$df[2]),4,dimnames = list(NULL,
  c("rep","t","l","yhat")))
```

```
#Create a table of interaction effects with i=1
inter2=matrix(int[1:16,4],4,4)


test=c(1,2,3,4)
par(mfrow=c(1,1))
plot(test,inter2[1,],type="n",ylab="Fitted",ylim=c(0,15),
xaxt = "n", #frame = TRUE,
xlab="Test",lty=1,lwd=2)    #,lab=c(4,5,7))
axis(1,at=c(1,2,3,4),labels=c("A","B","C","D"))
#axis(1, 1:4, LETTERS[1:4])
lines(test,inter2[1,],type="o",lty=1,lwd=2)
lines(test,inter2[2,],type="o",lty=2,lwd=2)
lines(test,inter2[3,],type="o",lty=3,lwd=2)
lines(test,inter2[4,],type="o",lty=4,lwd=2)
legend("topleft",c("Laundry","      1","      2","      3","      4"),lty=c(NA,1,2,
```

Some people use the program `lmer` from the package `lme4` to analyze split plot models. **The `lmer` program does not reproduce the analysis given here.** As of Feb. 10, 2023 `lmer` was producing incorrect sums of squares and mean squares for whole plot effects but *was giving the correct F tests*. The invidious thing is that it always gives $MS/F = MSE(2)$, suggesting that [if you assume the MS is correct], that all of the whole plot F tests are improperly using the subplot error term. This programming error is exacerbated by the fact that `lmer`'s ANOVA table fails to print out the Error terms, the source of the mistakes is not clear. The code book R-ALMIII illustrates the problem.

19.2 Split plots: abrasion resistance

Because these data are balanced, the `aov` command becomes a useful alternative to `lm` in that it can produce the proper ANOVA table. Unfortunately, `aov` does not work for unbalanced whole plot data as treated in the previous section.

```
rm(list = ls())
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-6.dat",
sep="",col.names=c("y","s","f","p","rep","rot"))
attach(abraid)
abraid
summary(abraid)

S=factor(s)
F=factor(f)
P=factor(p)
RT=factor(rot)
RP=factor(rep)
p2=p*p
bsfp <- aov(y ~ (S*F*P) + Error(RP:S:F:P) + RT + RT:(S*F*P))
summary(bsfp)
# Equivalent ways of specifying model
#bsfp <- aov(y ~ S*F*P*RT + Error(RP:S:F:P))
# Delete and retype the ^ in following!!!
#bsfp <- aov(y ~ (S+F+P)^3 + Error(RP:S:F:P) + RT + RT:(S+F+P)^3)
```

```
#bsfp <- aov(y ~ (S+F+P+RT)^3 + Error(RP:S:F:P) + RT:(S+F+P)^3)
```

The following uses `lm` to get all the pieces of the ANOVA table. Error 2 is the reported Error Error 1 is the effect `S:F:P:RP`; the output uses Error 2 when we need it to use Error 1. *You need to compute the whole plot F statistics yourself.*

```
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-6.dat",
sep="",col.names=c("y","s","f","p","rep","rot"))
attach(abraid)
abraid
summary(abraid)

S=factor(s)
F=factor(f)
P=factor(p)
RT=factor(rot)
RP=factor(rep)
p2=p*p
bsfp <- lm(y ~ (S+F+P+RT)^3 + S:F:P:RP + RT:(S+F+P)^3)
# Alternative model S*F*P*RT + S:F:P:RP + RT:(S*F*P)
anova(bsfp)
# Error 2 is the reported Error
# Error 1 is S:F:P:RP.
# All whole plot F tests and t tests use the wrong Error.


coef=summary(bsfp)
coef


#Error 2 residual plots

par(mfrow=c(2,1))
qqnorm(rstudent(bsfp),ylab="Standardized residuals")
plot(bsfp$fit,rstudent(bsfp),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")


# Wilk-Francia
rankit=qnorm(ppoints(rstandard(bsfp),a=I(3/8)))
ys=sort(rstandard(bsfp))
Wprime=(cor(rankit,ys))**2
Wprime


par(mfrow=c(2,2))
plot(s,rstudent(bsfp),xlab="Surface Treatments",
ylab="Standardized residuals",main="Residual-Surface Treatments plot")
plot(f,rstudent(bsfp),xlab="Fills",
ylab="Standardized residuals",main="Residual-Fills plot")
plot(p,rstudent(bsfp),xlab="Proportions",
ylab="Standardized residuals",main="Residual-Proportions plot")
plot(rot,rstudent(bsfp),xlab="Rotations",
ylab="Standardized residuals",main="Residual-Rotations plot")
```

19.2.1 Interaction plots

```
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-6.dat",
sep="",col.names=c("y","s","f","p","rep","rot"))
attach(abraid)
abraid
summary(abraid)

S=factor(s)
F=factor(f)
P=factor(p)
RT=factor(rot)
RP=factor(rep)
p2=p*p
r2=rot*rot
bsfp <- lm(y ~ S:F:P:RP + S:F:RT + p:rot + p:r2 + p2:rot + p2:r2)
anova(bsfp)
```

```
coef=summary(bsfp)
coef
```

```
ipy=bsfp$fit[s==1 & f==1 & rep==1]
ipp=p[s==1 & f==1 & rep==1]
iprot=rot[s==1 & f==1 & rep==1]
ip=c(ipp,iprot,ipy)
int=matrix(ip,9,3,dimnames = list(NULL,
  c("p","rot","yhat")))
int
```

```
#Create a table of interaction effects
inter2=matrix(int[1:9,3],3,3)
```

```
test=c(1,2,3)
plot(test,inter2[1,],type="n",ylab="Fitted",ylim=c(140,270),
xaxt = "n", #frame = TRUE,
xlab="Rotations",lty=1,lwd=2,lab=c(5,6,7))
axis(1,at=c(1,2,3),labels=c("1000","2000","3000"))
```

```
#axis(1, 1:4, LETTERS[1:4])
lines(test,inter2[1,],type="o",lty=1,lwd=2)
lines(test,inter2[2,],type="o",lty=2,lwd=2)
lines(test,inter2[3,],type="o",lty=3,lwd=2)
legend("bottomleft",c("Percent","25","50","75"),lty=c(NA,1,2,3))
```

19.2.2 Unbalanced subplots

The key is to have a separate effect for every whole plot (individual – in our example, piece of cloth) and then to model the subplot main effects and subplot by whole plot interactions as usual.

```
rm(list = ls())
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-6.dat",
sep="",col.names=c("y","s","f","p","rep","rot"))
attach(abraid)
abraid
summary(abraid)

S=factor(s)
F=factor(f)
P=factor(p)
RT=factor(rot)
RP=factor(rep)
p2=p*p
abwp <- lm(y ~ S:F:P:RP + RT + RT:(S+F+P)^3 )
coef=summary(abwp)
coef
anova(abwp)
```

I recently tried to interpret the ANOVA table that the above commands print out and it was not what I wanted, that is, it did not agree with the subplot analysis of Table 19.7. For example, `S:F:P:RP` had 14 degrees of freedom rather than the 23 I was expecting. Similarly, the interaction `S:RT` had 3 degrees of freedom rather than 2.

The code above *did give* the correct Error/Residuals line, which appears in both Tables 19.7 and 19.8. Similarly, the reduced models listed below give the appropriate error terms for Table 19.8. But do not trust anything other than the Error/Residuals line from these `anova` outputs

```
abwp <- lm(y ~ S:F:P:RP + RT + RT:(S+F+P)^2 )
anova(abwp)
abwp <- lm(y ~ S:F:P:RP + RT + RT:(S:F+S:P) )
anova(abwp)
abwp <- lm(y ~ S:F:P:RP + RT + RT:(S:F+F:P) )
anova(abwp)
abwp <- lm(y ~ S:F:P:RP + RT + RT:(S:P+F:P) )
anova(abwp)
abwp <- lm(y ~ S:F:P:RP + RT + RT:(P+S:F) )
anova(abwp)
abwp <- lm(y ~ S:F:P:RP + RT + RT:(S+F:P) )
anova(abwp)
abwp <- lm(y ~ S:F:P:RP + RT + RT:(F+S:P) )
```

```
anova(abwp)
abwp <- lm(y ~ S:F:P:RP + RT + RT:(F+S+P) )
anova(abwp)
```

There are no new skills involved in fitting the models of Tables 19.9, 19.10, and 19.11

19.2.3 Whole plot analysis with Error 1 residual plots

The whole plot analysis can be obtained either by summing over the three rotations or averaging over them. In the following code we read the data from an alternative data file that has separate columns (`yy1, yy2, yy3`) for each rotation. In this code we average over the 3 rotations. The sums of squares from the anova command are 1/3 of those in Table 19.7 but give exactly the same F tests. If we had used the sum of the rotations, the sums of squares would be 3 times greater than those in Table 19.7. The multiplicative factors 1/3 and 3 are based on the number of rotation levels that we have.

```
rm(list = ls())
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-6a.dat",
sep="",col.names=c("yy1","yy2","yy3","s","f","p","rep"))
attach(abraid)
abraid
summary(abraid)

yw=(yy1+yy2+yy3)/3
#Summary tables
S=factor(s)
F=factor(f)
P=factor(p)
p2=p*p
bsfp <- lm(yw ~ S:F:P-1)
coef=summary(bsfp)
# This prints the means for the 12 whole plot groups.
coef
# Get whole plot ANOVA table (almost)
bsfp <- lm(yw ~ S*F*P)
anova(bsfp)



# Whole Plot Residual Plots
par(mfrow=c(1,1))
qqnorm(rstudent(bsfp),ylab="Standardized residuals")
# Wilk-Francia
rankit=qnorm(ppoints(rstandard(bsfp),a=I(3/8)))
ys=sort(rstandard(bsfp))
Wprime=(cor(rankit,ys))**2
Wprime
par(mfrow=c(2,2))
plot(bsfp$fit,rstudent(bsfp),xlab="Fitted",
ylab="Standardized residuals",main="Residual-Fitted plot")
plot(s,rstudent(bsfp),xlab="Surface Treatmens",
ylab="Standardized residuals",main="Residual-Surface Treatments plot")
plot(f,rstudent(bsfp),xlab="Fills",
ylab="Standardized residuals",main="Residual-Fills plot")
plot(p,rstudent(bsfp),xlab="Proportions",
ylab="Standardized residuals",main="Residual-Proportions plot")
```

Table 19.12 contains the unbalanced data alternative to the standard ANOVA table along with fits that replace proportions effects with polynomials. There are no new skills involved. For the quadratic polynomial version of [SF][FP] (`S:F + F:P`) replace interaction `F:P` with regression terms `F:p + F:p2`.

The following code is for fitting models discussed after those discussed in Table 19.12.

```
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-6.dat",
sep="",col.names=c("y","s","f","p","rep","rot"))
attach(abraid)
abraid

S=factor(s)
F=factor(f)
P=factor(p)
p2=p*p
fb=f-1
fa=2-f
pa=p*fa
pa2=p*p*fa
pb=p*fb
pb2=p*p*fb
wsfp <- lm(y ~ S:F + pa + pa2 + pb +  pb2  -1)
anova(wsfp)
wsfp1 <- lm(y ~ S:F + pa + pa2 + pb  -1)
anova(wsfp1)
wsfp2 <- lm(y ~ S:F + pa + pa2  -1)
anova(wsfp2)
```

19.2.4 Final models

```
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-6.dat",
sep="",col.names=c("y","s","f","p","rep","rot"))
attach(abraid)
abraid

mtilde = 2*rot/rot
mtilde = mtilde - (rot<2)

mtildea=mtilde[f==1]
mtildeb=mtilde[f==2]

ya = y[f==1]
sa = s[f==1]
pa = p[f==1]
repa = rep[f==1]
```

```
rota = rot[f==1]

Mtildea=factor(mtildea)
Sa=factor(sa)
Pa=factor(pa)
RTa=factor(rota)
RPa=factor(repa)
p2a=pa*pa
r2a=rota*rota

bsfpa <- lm(ya ~ Sa + pa + RTa + Sa:Mtildea)
anova(bsfpa)
coefa=summary(bsfpa)
coefa
bsfpa$fit


yb = y[f==2]
sb = s[f==2]
pb = p[f==2]
repb = rep[f==2]
rotb = rot[f==2]

Mtildeb=factor(mtildeb)
Sb=factor(sb)
Pb=factor(pb)
RTb=factor(rotb)
RPb=factor(repb)
p2b=pa*pb
r2b=rotb*rotb

bsfpb <- lm(yb ~ Sb + RTb + Sb:Mtildeb)
anova(bsfpb)
coefb=summary(bsfpb)
coefb
bsfpb$fit
```

19.3 Multivariate ANOVA

The multivariate analysis presents terms that combine each whole plot term with its corresponding subplot interaction. It is a simple matter to separate those back out into whole plot terms and whole plot term by subplot term interactions. The whole plot terms will be equivalent to the corresponding split-plot whole plot analysis. The multivariate approach to looking at whole plot by subplot interactions will not be equivalent to the corresponding split-plot analysis. Since these issues are not discussed in the book, the R code is on my website in *R Code for ALM-III* at http://www.stat.unm.edu/~fletcher/R-ALMIII.pdf.

This illustration uses a different data file than that used for the split plot analysis. Near the end we illustrate how ro construct the necessary data from the split plot data file.

```
rm(list = ls())
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-6a.dat",
sep="",col.names=c("yy1","yy2","yy3","s","f","p","rep"))
attach(abraid)
abraid

y=cbind(yy1,yy2,yy3)
y           #y is a matrix with individual dependent variables as columns.

#Summary tables
S=factor(s)
F=factor(f)
P=factor(p)
p2=p*p
bsfp <- lm(y ~ S*F*P)
coef=summary(bsfp)
coef

anova(bsfp,test = c("Wilks"))
anova(bsfp,test = c("Roy"))
anova(bsfp,test = c("Hotelling-Lawley"))
anova(bsfp)       # Default is Pillai
anova(bsfp,test = c("Spherical"))   #Similar to split plot analysis
# try manova(y~S*F*P) and the summary thereof.

# Nicer output comes from the "car" package.
library(car)
bsfp.manova=Manova(bsfp)
summary(bsfp.manova)
# To obtain the E and H(S*F*P) matrices, H(S*F*P) is the 7th of the H matrices,
bsfp.manova$SSPE
bsfp.manova$SSP[7]




par(mfrow=c(2,2))
qqnorm(rstudent(bsfp)[,1],ylab="Standardized residuals y1")
qqnorm(rstudent(bsfp)[,2],ylab="Standardized residuals y2")
qqnorm(rstudent(bsfp)[,3],ylab="Standardized residuals y3")

par(mfrow=c(2,2))
plot(bsfp$fit[,1],rstudent(bsfp)[,1],xlab="Fitted y1",
ylab="Standardized residuals y1",main="Residual-Fitted plot")
plot(bsfp$fit[,2],rstudent(bsfp)[,2],xlab="Fitted y2",
ylab="Standardized residuals y2",main="Residual-Fitted plot")
plot(bsfp$fit[,3],rstudent(bsfp)[,3],xlab="Fitted y3",
ylab="Standardized residuals y3",main="Residual-Fitted plot")
```

Constructing the MANOVA variables from the split plot variables. Notice that I have changed the names of the effects being read in.

```
rm(list = ls())
abraid <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab19-6.dat",
sep="",col.names=c("yy","surf","fill","prop","Rep","rot"))
```

```
attach(abraid)
abraid

yy1=yy[rot==1]
yy2=yy[rot==2]
yy3=yy[rot==3]
s=surf[rot==1]
f=fill[rot==1]
p=prop[rot==1]
rep=Rep[rot==1]
```

For the effects, e.g., `f=fill[rot==3]` would have worked just as well.

19.4 Random effects and subsampling

19.4.1 Subsampling

19.4.2 Random Effects

Our computations involve only the use of `lm` but the R library `lme4` has a program `lmer` that deals with quite general random effects models. The `lmer` estimates agree with the `lm` ideas for balanced ANOVA problems but differ otherwise, cf. Christensen (2019). We illustrate both programs for the computations on the balanced one-way random effects model of Example 19.4.1.

```
rm(list = ls())
elec <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\EX19-4-1.dat",
sep="",col.names=c("y","S"))
attach(elec)
elec

Strip = factor(S)
elec.lm <- lm(y ~ Strip)
anova(elec.lm)

library(lme4)
elec.re <- lmer((y ~ (1 | Strip)), data = elec)
summary(elec.re)
```

```
anova(elec.lm)
          Df Sum Sq Mean Sq F value   Pr(>F)
Strip      3 10.873  3.6242  6.4539 0.002327 **
Residuals 24 13.477  0.5615
```

The estimate in the book is

$$\tilde{\sigma}_A^2 = \frac{MSTrts - MSE}{N} = \frac{3.6242 - 0.5615}{7} = 0.4375,$$

which is the Groups variance below. The Residual Variance below agrees with the Mean Sq Residuals above.

```
> summary(elec.re)
Random effects:
 Groups   Name        Variance Std.Dev.
 Strip    (Intercept) 0.4375   0.6615
 Residual             0.5615   0.7494
Number of obs: 28, groups:  Strip, 4
```

```
Fixed effects:
            Estimate Std. Error t value
(Intercept)  15.9964     0.3598   44.46
```

Chapter 20

Logistic Regression

20.1 Models for binomial data

20.2 Simple linear logistic regression

```
mice.sllr <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-1.dat",
  sep="",col.names=c("x","Ny","N","y"))
attach(mice.sllr)
mice.sllr
summary(mice.sllr)

#Summary tables
mi <- glm(y ~ x,family = binomial,weights=N)
mip=summary(mi)
mip
anova(mi)

rpearson=(y-mi$fit)/(mi$fit*(1-mi$fit)/N)^(.5)
rstand=rpearson/(1-hatvalues(mi))^(.5)
infv = c(y,mi$fit,hatvalues(mi),rpearson,
  rstand,cooks.distance(mi))
inf=matrix(infv,I(mip$df[1]+mip$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
# Note: delete y from table if it contains missing observations

#compute confidence intervals
R=mip$cov.unscaled
se <- sqrt(diag(R))
ci=c(mi$coef-qnorm(.975)*se, mi$coef+qnorm(.975)*se)
CI95 = matrix(ci,mip$df[1],2)
CI95
```

20.2.1 Goodness of fit tests

Don't use the Hosmer-Lemeshow chi-squared test. Only use the Pearson and Likelihood ratio tests if you have binomial data with all the N_is reasonably large!

20.2.2 Assessing predictive probabilities

If you want this stuff, use Minitab or SAS Proc Logistic.

20.2.3 Case diagnostics

These were provided with the earlier output.

20.3 Model testing

```
rm(list = ls())
mice.sllr <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-1.dat",
  sep="",col.names=c("x","Ny","N","y"))
attach(mice.sllr)
mice.sllr
summary(mice.sllr)

#Summary tables
mi <- glm(y ~ x,family = binomial,weights=N)
mip=summary(mi)
mip
mi3 <- glm(y ~ x+I(x^2)+I(x^3),family = binomial,weights=N)
mi3p=summary(mi3)
mi3p
anova(mi,mi3)
```

Alternatively,

```
mi3 = glm(y poly(x, degree = 3, raw=TRUE),family = binomial,weights=N)
```

will give exactly the same results whereas

`mi3 = glm(y poly(x, degree = 3),family = binomial,weights=N)` fits orthogonal polynomials which lead to the same model tests but provide different estimates.

20.4 Fitting logistic models

20.5 Binary data

```
rm(list = ls())
oring.sllr <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-3.dat",
  sep="",col.names=c("Case","Flt","y","s","x","no"))

attach(oring.sllr)
oring.sllr
#summary(oring.sllr)

#Summary tables
or <- glm(y ~ x,family = binomial)
orp=summary(or)
orp
anova(or)


#prediction
new = data.frame(x=c(31,53))
predict(or,new,type="response")
rpearson=(y-or$fit)/(or$fit*(1-or$fit))^(.5)
rstand=rpearson/(1-hatvalues(or))^(.5)
infv = c(y,or$fit,hatvalues(or),rpearson,
  rstand,cooks.distance(or))
inf=matrix(infv,I(orp$df[1]+orp$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
R2 = (cor(y,or$fit))**2
R2
```

20.5.1 Goodness of fit tests

Don't use these for binary data!

20.5.2 Case diagnostics

Given by the earlier code.

20.5.3 Assessing predictive probabilities

If you want things other than R^2, which was given earlier, use Minitab or SAS Proc Logistic.

20.6 Multiple logistic regression

Years ago software did not readily compute AIC so I used $A - q \equiv$ `AICq` because it was easy to compute by hand from the output df and G^2. In R, for a fitted model, say `svm`, the computation is `AICq=deviance(svm)-2*df.residual(svm)`. Now AIC is part of R's standard output and can be manipulated as `AIC(svm)`. **The key point to notice is that, for various models, differences between R's AIC agree with the corresponding differences between** $A - q$**,** so they lead to the same ordering of models.

```
rm(list = ls())
chap.mlr <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\chapman.dat",
  sep="",col.names=c("Case","Ag","S","D","Ch","H","W","y"))

attach(chap.mlr)
chap.mlr
#summary(chap.mlr)

#Summary tables
cm <- glm(y ~ Ag+S+D+Ch+H+W,family = binomial)
cmp=summary(cm)
cmp
anova(cm)

# Diagnostics
rpearson=(y-cm$fit)/(cm$fit*(1-cm$fit))^(.5)
rstand=rpearson/(1-hatvalues(cm))^(.5)
infv = c(y,cm$fit,hatvalues(cm),rpearson,
  rstand,cooks.distance(cm))
inf=matrix(infv,I(cmp$df[1]+cmp$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
```

20.6.1 Best subset logistic regression

This isn't in the book.

Below is the method discussed in Christensen (1997, Section 4.4, *Log-linear Models and Logistic Regression*). The method starts with the full model and performs only one step of the Newton-Raphson/Iteratively Reweighted Least Squares algorithm to determine the best models. This is a far better procedure than the score test method used by SAS Proc Logistic because it starts from the full model, which should be a good model, rather than the intercept only model used by the score test.

```
rm(list = ls())
chap <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\chapman.dat",
  sep="",col.names=c("Case","Ag","S","D","Ch","H","W","y"))
attach(chap)
chap
summary(chap)

#Summary tables
ch = glm(y ~ Ag+S+D+Ch+H+W,family=binomial)
chp=summary(ch)
chp
#anova(ch)

rwt=ch$fit*(1-ch$fit)
yy=log(ch$fit/(1-ch$fit))+(y-ch$fit)/rwt
# If Bin(n_i,p_i)s have n_i different from 1, multiply
# rwt and second term in yy by by n_i

ch1 <- lm(yy ~ Ag+S+D+Ch+H+W,weights=rwt)
ch1p=summary(ch1)
ch1p
anova(ch1)
# Note the agreement between the glm and lm fits!!!


#install.packages("leaps")
library(leaps)
x <- model.matrix(ch1)[,-1]
g <- regsubsets(x,yy,nbest=3,weights=rwt)

# assign number of best models and number of predictor variables.
nb=3
xp=ch1p$df[1]-1
dfe=length(y)- 1- c(rep(1:(xp-1),each=nb),xp)
g <- regsubsets(x,yy,nbest=nb,weights=rtw)
gg = summary(g)
tt=c(gg$rsq,gg$adjr2,gg$cp,sqrt(gg$rss/dfe))
tt1=matrix(tt,nb*(xp-1)+1,4,
 dimnames = list(NULL,c("R2", "AdjR2", "Cp", "RootMSE")))
tab1=data.frame(tt1,gg$outmat)
tab1
```

Another possible source for best subset logistic regression is the package/program `bestglm` which seems to do full, rather than one-step, fits of the models, cf. also the `glmulti` package/program of Calcagno and de Mazancourt (2010). The package `StepReg` has a program `stepwiseLogit` that behaves similar to their `stepwise` for linear models. I would be worried about whether the selection="score" option performs one-step approximations based on the intercept only model rather (like SAS) than the full model as illustrated here.

Calcagno, Vincent and de Mazancourt, Claire (2010). glmulti: An R Package for Easy Automated Model Selection with (Generalized) Linear Models, *Journal of Statistical Software*, Volume 34, Issue 12.

20.6.2 Stepwise logistic regression

This chooses models based on the AIC criterion. As illustrated earlier, obtain the `glm` output.

```
ch = glm(y ~ Ag+S+D+Ch+H+W,family=binomial)
chstep <- step(ch, direction="backward")
chstep
```

Other "directions" include `both` and `forward` but forward requires additional commands, see Section 10.3. You get similar results by replacing the `glm` output in `ch` with the `lm` output from `ch1 = lm(yy   Ag+S+D+Ch+H+W,weights=rtw)`.

Haven't run yet. The package `StepReg`'s `stepwiseLogit` program allows stepping based on P values of G^2, e.g., step(ch, selection="backward",select="SL",sls=0.05,sigMethod="LRT"). I would guess that when `sls` is specified, telling it to use significance levels,`select="SL"`, would be redundant but I haven't checked. In its description it says that it allows Wald $[\hat{\beta}/\text{SE}(\hat{\beta})]$ and score tests but the options are for G^2 and score tests.

20.7 ANOVA models

```
tense <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10.dat",
  sep="",col.names=c("High","Low","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
M=factor(Ms)
D=factor(Dr)
T=cbind(High,Low)
ts <- glm(T ~ W*M + M*D,family = binomial)
tsp=summary(ts)
tsp
anova(ts)
```

20.8 Ordered categories

```
rm(list = ls())
abop <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-15.dat",
  sep="",col.names=c("Case","Race","Sex","Age","Yes","No","Total"))
```

```
attach(abop)
abop
#summary(abop)

#Summary tables
R=factor(Race)
S=factor(Sex)
A=factor(Age)
y=Yes/Total
# Model (20.8.1)
ab <- glm(y ˜ R:S + A,family = binomial,weights=Total)
abp=summary(ab)
abp

odds=ab$fit/(1-ab$fit)
odds

# Model (20.8.2)
ab <- glm(y ˜ R:S + Age,family = binomial,weights=Total)
abp=summary(ab)
abp

# Model (20.8.3)
Men=Race*(Sex-1)
m=factor(Men)
ab <- glm(y ˜ m + A,family = binomial,weights=Total)
abp=summary(ab)
abp

# Model (20.8.4)
ab <- glm(y ˜ m + Age,family = binomial,weights=Total)
abp=summary(ab)
abp

odds=ab$fit/(1-ab$fit)
oddstable=matrix(odds,6,4,dimnames = list(NULL,
     c("Male", "   White Female", "      Male","   Nonwhite Female")))
oddstable
```

Chapter 21

Log-Linear Models

The generalized linear model procedure `glm` uses Newton-Raphson (iteratively reweighted least squares)? [The output refers to it as Fisher Scoring.] To use iterative proportional fitting use

```
library(MASS)
loglm(y ~ model)
```

Within R, the likelihood ratio chi-square G^2 is labeled as the *residual deviance*.

21.1 Models for two-factor tables

We now use log-linear models to reproduce results from the Chapter 5.

```
rm(list = ls())
occ <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab21-3.dat",
  sep="",col.names=c("y","R","O","RCP","Q"))
#  Early versions have tab21-2.dat identical to tab21-3.dat
#  wherein tab21-2.dat has two unnecessary columns.
# Until the next subsection, ignore the variables RCP and Q!
attach(occ)
occ
#summary(occ)

#Summary tables
r=factor(R)
o=factor(O)
ts <- glm(y ~ r + o,family = poisson)
tsp=summary(ts)
tsp
anova(ts)


rpearson=(y-ts$fit)/(ts$fit)^(.5)
rstand=rpearson/(1-hatvalues(ts))^(.5)
infv = c(y,ts$fit,hatvalues(ts),rpearson,
  rstand,cooks.distance(ts))
inf=matrix(infv,I(tsp$df[1]+tsp$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf

#prediction
#new = data.frame(r=c("1"),o=c("1"))
#predict(ts,new,type="response")
```

21.1.1 Lancaster-Irwin Partitioning

We begin by reproducing the independence/homogeneity analysis for the full table using two new variables one of which collapses the Roman Catholics and Protestants but identifies the other variables and another that identifies Catholics and Protestants but collapses the other variables. In this example, there is only one other variable, so nothing to collapse.

```
rm(list = ls())
occ <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab21-3.dat",
  sep="",col.names=c("y","R","O","RCP","Q"))
#  RCP lumps Roman Catholics and Protestants together
#  Q distinguishes between Roman Catholics and Protestants
attach(occ)
occ
#summary(occ)

#Reproduce the independence/homogeneity analysis using RCP and Q

r=factor(R)
o=factor(O)
rcp=factor(RCP)
q=factor(Q)
ts2 <- glm(y ~ o + rcp:q,family = poisson)
tsp2=summary(ts2)
tsp2


rpearson=(y-ts2$fit)/(ts2$fit)^(.5)
rstand=rpearson/(1-hatvalues(ts2))^(.5)
infv = c(y,ts2$fit,hatvalues(ts2),rpearson,
  rstand,cooks.distance(ts2))
inf=matrix(infv,I(tsp2$df[1]+tsp2$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
```

You can check to see that `y`, `yhat`, and `Pearson` all agree with the earlier analysis.

We now fit a model to the entire data that fits independence/homegeneity to the reduced table while fitting the other cells perfectly. The G^2 (residual deviance) for this model is that for the reduced model.

```
ts3 <- glm(y ~ o:rcp + rcp:q,family = poisson)
tsp3=summary(ts3)
tsp3

rpearson=(y-ts3$fit)/(ts3$fit)^(.5)
rstand=rpearson/(1-hatvalues(ts3))^(.5)
infv = c(y,ts3$fit,hatvalues(ts3),rpearson,
  rstand,cooks.distance(ts3))
inf=matrix(infv,I(tsp3$df[1]+tsp3$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
anova(ts2,ts3)
```

The difference between the full model G^2 and the reduced model G^2 gives the G^2 for the collapsed table. The degrees of freedom are also found by subtraction. The `anova` command gives all three of the relevant test statistics.

21.2 Models for three-factor tables

21.2.1 Testing models

EXAMPLE 21.2.1

```
rm(list = ls())
per <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab21-1.dat",
  sep="",col.names=c("P","C","B","y"))

attach(per)
per
#summary(per)
p=factor(P)
c=factor(C)
b=factor(B)
m7 <- glm(y ~ p:c + p:b + c:b,family = poisson)
m7s=summary(m7)
m6 <- glm(y ~ p:c + p:b,family = poisson)
m6s=summary(m6)
m5 <- glm(y ~ p:c + c:b,family = poisson)
m5s=summary(m5)
m4 <- glm(y ~ p:b + c:b,family = poisson)
m4s=summary(m4)
m3 <- glm(y ~ p + c:b,family = poisson)
m3s=summary(m3)
m2 <- glm(y ~ c + p:b,family = poisson)
m2s=summary(m2)
m1 <- glm(y ~ p:c + b,family = poisson)
m1s=summary(m1)
m0 <- glm(y ~ p + c + b,family = poisson)
m0s=summary(m0)
df=c(m7$df.residual,m6$df.residual,m5$df.residual,m4$df.residual,
     m3$df.residual,m2$df.residual,m1$df.residual,m0$df.residual)
G2=c(m7$deviance,m6$deviance,m5$deviance,m4$deviance,
     m3$deviance,m2$deviance,m1$deviance,m0$deviance)
A2q=G2-(2*df)
modelm=c(df,G2,A2q)
model=matrix(modelm,8,3,dimnames = list(NULL,c("df", "G2", "A2q")))
model
```

21.3 Estimation and odds ratios

EXAMPLE 21.3.1 Auto accident injuries.

```
rm(list = ls())
sb <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\EX21-3-1.dat",
  sep="",col.names=c("nuLL","i","j","k","y"))

attach(sb)
```

```
per
#summary(sb)
I=factor(i)
J=factor(j)
K=factor(k)
m7 <- glm(y ~ I:J + I:K + J:K,family = poisson)
m7s=summary(m7)

rpearson=(y-m7$fit)/(m7$fit)^(.5)
rstand=rpearson/(1-hatvalues(m7))^(.5)
infv = c(y,m7$fit,hatvalues(m7),rpearson,rstand,
     cooks.distance(m7))
inf=matrix(infv,I(m7s$df[1]+m7s$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
```

EXAMPLE 21.3.2 Classroom behavior.

```
rm(list = ls())
sb <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\EX21-3-2.dat",
  sep="",col.names=c("y","i","j","k"))

attach(sb)
per
#summary(sb)
I=factor(i)
J=factor(j)
K=factor(k)
m7 <- glm(y ~ I + J:K,family = poisson)
m7s=summary(m7)

rpearson=(y-m7$fit)/(m7$fit)^(.5)
rstand=rpearson/(1-hatvalues(m7))^(.5)
infv = c(y,m7$fit,hatvalues(m7),rpearson,rstand,
     cooks.distance(m7))
inf=matrix(infv,I(m7s$df[1]+m7s$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
```

21.4 Higher dimensional tables

Muscle tension changes.

```
rm(list = ls())
tense <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10a.dat",
  sep="",col.names=c("y","Tn","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
```

```
M=factor(Ms)
D=factor(Dr)
T=factor(Tn)
m7 <- glm(y ~ T:W:M + T:W:D + T:M:D + W:M:D,family = poisson)
m4 <- glm(y ~ T:W + T:M + T:D + W:M + W:D + M:D,family = poisson)
m0 <- glm(y ~ T + W + M + D,family = poisson)

df=c(m7$df.residual,m4$df.residual,m0$df.residual)
G2=c(m7$deviance,m4$deviance,m0$deviance)
A2q=G2-(2*df)
modelm=c(df,G2,A2q)
model=matrix(modelm,3,3,dimnames = list(NULL,c("df", "G2", "A2q")))
model
```

You can also get the key statistics from the following commands

```
m7 <- glm(y ~ T*W*M + T*W*D + T*M*D + W*M*D,family = poisson)
m7p=summary(m7)
m7p
anova(m7)
```

What you want is in the last 2 columns. R is fitting the models sequentially, adding in each term on the left.

21.5 Ordered categories

EXAMPLE 21.5.1

```
rm(list = ls())
abt <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\EX21-5-1.DAT",
  sep="",col.names=c("c","p","y"))
attach(abt)
abt


C=factor(c)
P=factor(p)
m3 <- glm(y ~ C + P + C:p,family = poisson)   #[C][P][C_1]
m2 <- glm(y ~ C + P + c:P,family = poisson)   #[C][P][P_1]
m1 <- glm(y ~ C + P + c:p,family = poisson)   #[C][P][gamma]
m0 <- glm(y ~ C + P,family = poisson)         #[C][P]
df=c(m3$df.residual,m2$df.residual,m1$df.residual,m0$df.residual)
G2=c(m3$deviance,m2$deviance,m1$deviance,m0$deviance)
A2q=G2-(2*df)
modelm=c(df,G2,A2q)
model=matrix(modelm,4,3,dimnames = list(NULL,c("df", "G2", "A2q")))
model


m1s=summary(m1)
m1s
anova(m1)

rpearson=(y-m1$fit)/(m1$fit)^(.5)
```

```
rstand=rpearson/(1-hatvalues(m1))^(.5)
infv = c(y,m1$fit,hatvalues(m1),rpearson,rstand,
     cooks.distance(m1))
inf=matrix(infv,I(m1s$df[1]+m1s$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
```

EXAMPLE 21.5.2

```
rm(list = ls())
abt <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-4.DAT",
  sep="",col.names=c("R","S","A","O","y"))
attach(abt)
abt
#summary(abt)

r=factor(R)
o=factor(O)
s=factor(S)
a=factor(A)
A2=A*A
#[RSO][OA]
ab <- glm(y ~ r:s:o + o:a ,family = poisson)
abp=summary(ab)
abp
anova(ab)

#[RSO][A][O_1][O_2]
ab2 <- glm(y ~ r:s:o + a + o:A + o:A2,family = poisson)
abp2=summary(ab2)
abp2
anova(ab2)

#[RSO][A][O_1]
ab3 <- glm(y ~ r:s:o + a + o:A ,family = poisson)
abp3=summary(ab3)
abp3
anova(ab3)

rpearson=(y-ab3$fit)/(ab3$fit)^(.5)
rstand=rpearson/(1-hatvalues(ab3))^(.5)
infv = c(y,ab3$fit,hatvalues(ab3),rpearson,
  rstand,cooks.distance(ab3))
inf=matrix(infv,I(abp3$df[1]+abp3$df[2]),6,dimnames = list(NULL,
   c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
```

21.6 Offsets

```
bis <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-5.DAT",
  sep="",col.names=c("index","l","y"))
attach(bis)
```

```
bis
ll=log(l)
ab2 <- glm(y ~ ll ,family = poisson)
abp2=summary(ab2)
abp2
ab3 <- glm(y ~ offset(ll) ,family = poisson)
abp3=summary(ab3)
abp3
anova(ab3)
```

21.7 Relation to logistic models

There is no computing associated with this section.

21.8 Multinomial responses

21.9 Logistic discrimination and allocation

The first thing we have to do is create the 3×21 table illustrated in the book. We then fit the model and finally we get the entries for the book's Tables 21.11 and 21.12.

```
rm(list = ls())
cush <- read.table("C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-11.DAT",
  sep="",col.names=c("Syn","Tetra","Preg"))
attach(cush)
cush

#Create a 3 x 21 table of 0-1 entries,
#each row has 1's for a different type of syndrome
j=rep(seq(1,21),3)
i=c(rep(1,21),rep(2,21),rep(3,21))
Tet=c(Tetra,Tetra,Tetra)
Pre=c(Preg,Preg,Preg)
y=c(Syn,Syn,Syn)
y[1:6]=1
y[7:21]=0
y[22:27]=0
y[28:37]=1
y[38:58]=0
y[59:63]=1
datal=c(y,i,j,Tet,Pre)
datl=matrix(datal,63,5,dimnames = list(NULL,c("y", "i", "j","Tet","Pre")))
datl

#Fit the log-linear model for logistic discrimination.
i=factor(i)
j=factor(j)
lp=log(Pre)
lt=log(Tet)
ld <- glm(y ~ i + j + i:lt +i:lp ,family = poisson)
ldp=summary(ld)
ldp
anova(ld)
```

```
# Table 21.12
q=ld$fit
# Divide by sample sizes
p1=ld$fit[1:21]/6
p2=ld$fit[22:42]/10
p3=ld$fit[43:63]/5
# Produce table
estprob = c(Syn,p1,p2,p3)
EstProb=matrix(estprob,21,4,dimnames = list(NULL,c("Group", "A", "B","C")))
EstProb

# Table 21.13 Proportional prior probabilities.
post = c(Syn,ld$fit)
PropProb=matrix(post,21,4,dimnames = list(NULL,c("Group", "A", "B","C")))
PropProb

# Table 21.13 Equal prior probabilities.
p=p1+p2+p3
pp1=p1/p
pp2=p2/p
pp3=p3/p
post = c(Syn,pp1,pp2,pp3)
EqProb=matrix(post,21,4,dimnames = list(NULL,c("Group", "A", "B","C")))
EqProb
```

Chapter 22

Exponential and Gamma Regression: Time to Event Data

22.1 Exponential regression

```
fz <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab22-1.dat",
sep="",col.names=c("y","WBC","AG"))
attach(fz)
fz
#summary(fz)


a=factor(AG)
lw=log10(WBC)
er <- glm(y~a+lw-1, family=Gamma(link="log"))
erp=summary(er,dispersion=1)
erp
anova(er)
mpar(2,2)
plot(er)


confit(er)
#compute confidence intervals
R=erp$cov.unscaled
se <- sqrt(diag(R)) * erp$sigma
ci = c(er$coef - qt(.975,erp$df[2])*se,er$coef + qt(.975,erp$df[2])*se)
CI95 = matrix(ci,erp$df[1],2)
CI95

#Test nu1=2nu2

a=factor(AG)
NAG=3-AG
lw=log10(WBC)
er <- glm(y~NAG+lw:a-1, family=Gamma(link="log"))
erp=summary(er,dispersion=1)
erp

#COMPUTING ISSUES
```

```
a=AG-1
lw=log10(WBC)
er <- glm(y~a+lw+lw:a, family=Gamma(link="log"))
erp=summary(er,dispersion=1)
erp
```

22.2 Gamma regression

```
fz <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab22-1.dat",
sep="",col.names=c("y","WBC","AG"))
attach(fz)
fz
#summary(fz)


#Summary tables
a=factor(AG)
lw=log10(WBC)
er <- glm(y~a+lw:a-1, family=Gamma(link="log"))
erp=summary(er)
erp


anova(er)
plot(er)

#COMPUTING ISSUES

a=AG-1

er <- glm(y~a+lw+lw:a, family=Gamma(link="log"))
erp=summary(er)
erp
```

22.2.1 Offsets

Test ACOVA slope of 1.

```
fz <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab22-1.dat",
sep="",col.names=c("y","WBC","AG"))
attach(fz)
fz
#summary(fz)


a=factor(AG)
lw=log10(WBC)
er <- glm(y~a+offset(-1*lw), family=Gamma(link="log"))
erp=summary(er)
```

```
erp


anova(er)
```

Chapter 23

Nonlinear Regression

See also `nlstools`.

23.1 Using nls

```
rm(list = ls())
bz <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab23-1.dat",
sep="",col.names=c("x1","x2","T","x4","r"))
attach(bz)
bz
#summary(bz)

y <- 100/r
x3=(1/T)-(1/648)


mlf1 <- nls(y ~ exp(b0 +  b1*x3)  *(1/x2) +  exp( b2 +  b3*x3)  *(x4/x1),
start=list(b0 = 1.3130, b1 = 11908, b2 = -0.23463, b3 = 10559.5),
trace=T)
mlf1p <- summary(mlf1)
mlf1p
confint(mlf1, level=0.95)



mlf1 <- nls(y ~ exp(b0 +  b1*x3)  *(1/x2) +  exp( b2 +  b3*x3)  *(x4/x1),
start=list(b0 = 1, b1 = 10000, b2 = 1, b3 = 10000),
trace=T)
mlf1p <- summary(mlf1)
mlf1p
confint(mlf1, level=0.95)
```

23.2 Linearized models

```
rm(list = ls())
bz <- read.table("C:\\E-drive\\Books\\ANREG2\\NewData\\tab23-1.dat",
sep="",col.names=c("x1","x2","T","x4","r"))
attach(bz)
```

```
bz
#summary(bz)

y <- 100/r
x3=(1/T)-(1/648)
b0 <- 1.3130
b1 <- 11908
b2 <- -0.23463
b3 <- 10559.5

Fb <- exp(b0 +  b1*x3)  *(1/x2) +  exp( b2 +  b3*x3)  *(x4/x1)
dFb0 <- exp(b0 +  b1*x3)  *(1/x2)
dFb1 <- exp(b0 +  b1*x3)  *(x3/x2)
dFb2 <-   exp( b2 +  b3*x3) *(x4/x1)
dFb3 <-   exp( b2 +  b3*x3)  *x3*(x4/x1)
Zb = b0 * dFb0 + b1 * dFb1 + b2 * dFb2 + b3 * dFb3
yy = y - Fb + Zb
y
Fb
Zb
yy
mlf = lm(yy ~ dFb0 + dFb1 + dFb2 + dFb3 - 1)
zstar= model.matrix(mlf)
zstar
mlfp <- summary(mlf)
mlfp
confint(mlf, level=0.95)




qqnorm(rstandard(mlf),ylab="Standardized residuals")
rankit=qnorm(ppoints(rstandard(mlf),a=I(3/8)))
ys=sort(rstandard(mlf))
Wprime=(cor(rankit,ys))**2
Wprime


par(mfrow=c(2,1))
plot(mlf$fit,rstandard(mlf),xlab="dhat",
ylab="Standardized residuals",main="Residual-dhat plot")
plot(Fb,rstandard(mlf),xlab="yhat",
ylab="Standardized residuals",main="Residual-yhat plot")


par(mfrow=c(2,2))
plot(x1,rstandard(mlf),xlab="x1",ylab="Standardized residuals",
     main="Residual-x1 plot")
plot(x2,rstandard(mlf),xlab="x2",ylab="Standardized residuals",
     main="Residual-x2 plot")
plot(T,rstandard(mlf),xlab="x3",ylab="Standardized residuals",
     main="Residual-T plot")
plot(x4,rstandard(mlf),xlab="x4",ylab="Standardized residuals",
     main="Residual-x4 plot")


infv = c(y-Fb,mlf$fit,hatvalues(mlf),rstandard(mlf),rstudent(mlf),
     cooks.distance(mlf))
inf=matrix(infv,I(mlfp$df[1]+mlfp$df[2]),6,dimnames = list(NULL,
  c("y", "yhat", "lev","r","t","C")))
inf





index = seq(1,54)
par(mfrow=c(2,1))
plot(index,hatvalues(mlf),xlab="Index",
     ylab="Leverage",main="Leverage plot")
plot(index,cooks.distance(mlf),xlab="Index",
     ylab="Cook’s Distance",main="Cook’s distance plot")
```

Appendix A: Matrices and Vectors

Commands for matrix algebra where given at the beginning of Chapter 11.

A.8 Eigenvalues and vectors

Eigenvalues and eigenvectors were discussed in Appendix Section A.8. We now perform principal component regression using `eigen`.

```
Z=coleman[,2:6]                     #Define predictor matrix from data frame
e <- eigen(cor(Z),symmetric=TRUE)
e$values
e$vectors
Zs <- scale(Z)                              #Centers and scales the matrix Z
PC <- Zs %*% e$vec                     #Matrix of principal component scores
co <- lm(y ~ PC)
cop = summary(co)
cop
anova(co)
```

Proposition A.8.2 presented the singular value decomposition. The presentation was for a matrix A that was symmetric however the R function works for nonsymmetric matrices. When A is symmetric, `P = PP` below. The following code reproduces Example A.8.3.

```
A = matrix(c(3,1,-1,1,3,-1,-1,-1,5),ncol=3)
dc <- svd(A)
phi <- dc$d
phi
D <- diag(phi)
P <- dc$u
P
PP <- dc$v
PP
P %*% D %*% t(PP) #  A = P D P'
t(PP) %*% A %*% P #  D = P' A P
```

As in the book, the eigenvalues in `phi` are 6, 3, 2. The columns of `P` and `PP` are decimal representations of those in the book.

Chapter 24

More Stuff

```
oakes <- read.table("C:\\E-drive\\AmericanStatistician\\oakes.dat",
  sep="",col.names=c("n","i","j"))
attach(oakes)
oakes
#summary(oaks)

#Summary tables
I=factor(i)
J=factor(j)
oak <- glm(n ~ I + J + I*j,family = poisson)
oak
anova(oak)
```

Index