Ronald Christensen
Department of Mathematics and Statistics
University of New Mexico

Preliminary Version of R Commands for – Statistical Learning: A Second Course in Regression

Springer

Preface

This online book is an R companion to *Statistical Learning: A Second Course in Regression* (*SL*). *SL* was largely compiled from material in Christensen (2015), *Analysis of Variance, Design, and Regression: Linear Models for Unbalanced Data (ANREG-II)*, Christensen (2020), *Plane Answers to Complex Questions: The Theory of Linear Models (PA or for the fifth edition PA-V)* and Christensen (2019) *Advanced Linear Modeling: Statistical Learning and Dependent Data (ALM-III)*. Similarly this computer code book was largely compiled from the R code for the first and last of those books: *R Commands for — Analysis of Variance, Design, and Regression: Linear Modeling of Unbalanced Data* (`http://www.stat.unm.edu/~fletcher/Rcode.pdf`) and *R Commands for — Advanced Linear Modeling III* (`http://www.stat.unm.edu/~fletcher/R-ALMIII.pdf`). The data files used in these programs are data files that correspond either to *ANREG-II* or *ALM-III* and can be downloaded from `https://www.stat.unm.edu/~fletcher/newavdr_data.zip` or `https://www.stat.unm.edu/~fletcher/ALM-III-DATA.zip`, respectively. The data files for *SL* can also be downloaded directly from `https://www.stat.unm.edu/~fletcher/SL-Data.zip` with the data in Table x.y appearing in file SLx-y.dat.

This book presupposes that the reader is already familiar with downloading R, plotting data, reading data files, transforming data, basic housekeeping, loading R packages, and specifying basic linear models. That is the material in Chapters 1 and 3 of the R commands for *ANREG-II*. 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`.
An overview of packages in R for statistical learning is available at `https://cran.r-project.org/web/views/MachineLearning.html` and `https://CRAN.R-project.org/view=MachineLearning`.

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", i.e.,

```
~
```

will often copy incorrectly so that **you may need to delete the copied version of tilde and retype it**.

Programming is not of great interest to me. I did not even bother to learn R until the late aughts and only then because I was editing an appendix on R largely written by Adam Branscum. (Thank you Adam.) 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 the particular task of writing code for this book.

Ronald Christensen
Albuquerque, New Mexico
March, 2019

List of libraries used (so you can install them all at once). Install all the packages indicated in `http://www.stat.unm.edu/~fletcher/Rcode.pdf` and `http://www.stat.unm.edu/~fletcher/R-ALMIII.pdf` as well as

```
install.packages("neuralnet")
```

Contents

Chapter 1
Linear Regression

1.1 An Example

In most chapters the section structure will reflect the section structure of the corresponding chapter in *SL*. As Chapter 1 is just a review of regression, that seems unnecessary.

I find that the easiest way to program things is to find some initial code that works and modify it. The code given below generates the basic results for analyzing the Coleman Report data from the book. It starts by reading in the data and printing it out in order to verify that it has been read correctly.

The `#Summary tables` section of the code starts the analysis by using the `lm` command to fit the linear regression. It then prints out the table of coefficients and a table that gives all the pieces of the standard three line ANOVA table. It then gives another ANOVA table that contains the sequential sums of squares. Finally it prints out confidence intervals for the regression coefficients. The middle section `#Predictions` gives both the confidence interval for a point on the regression surface and the corresponding prediction interval. The last section `#Diagnostics` produces a table of diagnostic quantities that includes the observations, fitted values, standardized and *t* residuals, and Cook's distances. It then produces the normal plot of the standardized residuals with the Sharpiro-Francia statistic W' and the plot of the standardized residuals against the fitted values.

```
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(lm(y~1),co)
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")

#Diagnostics
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")
```

1.2 Inferential Procedures

See Section 1.

1.3 General Statement of Model

No computing.

1.4 Regression Surfaces and Prediction

See Section 1.

1.5 Comparing Regression Models

In Section 1.5 of the book we perform F tests to compare two specific models. The results 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.

1.6 Sequential Fitting

See Section 1.

1.7 Reduced Models and Prediction

No computing.

1.8 Collinearity

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, if we form a matrix of predictor variables `Z`, we can get them all at once

```
co = lm(y ~ x1+x2+x3+x4+x5)
Z <- model.matrix(co)[,-1]
cor(y,Z)}.
```

Similarly, the correlations given in Example 1.8.1 are produced by `cor(Z)`.

1.9 More on Model Testing

This section is about turning hypotheses into reduced models, many of which involve an offset term. The code is just for reading in the new data and fitting the reduced models.

```
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 (1.9.1)
m1=lm(y~x1+x2+x3+x4+x5)
summary(m1)
anova(m1)

#Model (1.9.2)
bsum=x2+x3
m2=lm(y~x1+bsum+x4+x5)
summary(m2)
anova(m2)

#Model (1.9.3)
bdif=x2-x3
m3=lm(y~x1+bdif+x4+x5)
summary(m3)
anova(m3)

#Model (1.9.4)
bsum=x2+x3
m4=lm(y~offset(.5*x3) + x1+bsum+x4+x5)
summary(m4)
anova(m4)

#Model (1.9.5)
```

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

#Model (1.9.6)
m6=lm((y-3.5*x5) ~ x1+x2+x3+x4)
summary(m6)
anova(m6)

#Model (1.9.7)
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)
```

1.10 Diagnostics

This section looks at the diagnostic quantities produced earlier but it also goes on to look at refitting models after deleting some observations. To delete cases 18 and 3 use the following code. Then rerun the earlier code except the diagnostic table needs to be modified.

```
# Read data.
y[18]=NA
y[3]=NA
# Fit model here
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
```

I'm not sure I have actually fixed the diagnostic table.

1.11 Other Useful Tools

ANREG-II, Chapter 7 discusses transformations to deal with nonnormality and heteroscedasticity with R code in Chapter 7 of its online code book. In addition, the package `library(lmtest)` includes versions of the Durbin-Watson test for serial correlation and the Breusch-Pagan/Cook-Weisberg test for heteroscedasticity. Both are introduced in *PA-V*, Chapter 12.

I don't remember why I have these commands here other than that they have some unusual syntax worth noting. The last command is very useful in that it will center the variable(s) (subtract the means) and/or rescale them variable. It also works on the columns of matrices.

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

Chapter 2
Matrix Formulation

In this chapter we use R's matrix methods to produce results for multiple regression. 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.

2.1 Matrix Algebra

This section contains the computing commands that go along with Appendix A except for those associated with the eigenvalues and eigenvectors, cf. Section A.8.

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(A)                      # diag. vector from matrix A
diag(v,nrow=length(v))       # diag. matrix from vector v
%*%                          # 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=T,scale=T)    # subtract col means, divide by col std. dev
rankMatrix(X)                # Compute r(X). Can be dicey.
```

2.2 Matrix Formulation of Models

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.

2.3 Least Squares Estimation

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
Bhat
# library(MASS) allows a more general method
# 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.

2.4 Inference

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.

2.5 Diagnostics

The book discusses how to compute two of our standard diagnostic quantities: the leverages and the standardized residuals. *PA* 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.

2.6 Basic Notation and Concepts

Nothing here.

2.7 Weighted Least Squares

This section needs checking. As with most programs for regression, weights can be specified within the program. In `lm` you specify a vector `wt`. In our example we assume a weighting vector `wts` has been specified.

```
# Fit WLS model
wlsfit <- lm(y ~ X, wt=wts)
summary(wlsfit)
anova(lm(y ~ 1,wt=wts),wlsfit)
```

2.7.1 Generalized Least Squares

This section discusses an example of fitting models with a **nondiagonal weighting matrix**. **It still needs work.**

Consider a random walk. Start with independent variables $w_1, \dots, w_{10}$ with $\mathrm{E}(w_j) = \mu$ and $\mathrm{Var}(w_j) = \sigma^2$. From these define $y_1 = w_1$ and $y_i = w_1 + \cdots + w_i$. Now suppose you want to estimate μ. If you can see the w_i, you just average them. If you see the y_is, you can reconstruct the w_is as $w_1 = y_1$, $w_2 = y_2 - y_1$, $w_3 = y_3 - y_2$, etc. It is easy to see that $\bar{w}_{\cdot} = y_{10}/10$. Linear model theory will give the same estimate.

$$\mathrm{E}(y_i) = i\mu; \quad \mathrm{Var}(y_i) = i\sigma^2; \mathrm{Cov}(y_i, y_{i'}) = \min(i, i')\sigma^2,$$

so with $Y = (y_1, \dots, y_{10})'$

$$\mathrm{E}(Y) = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 10 \end{bmatrix} \mu; \qquad \mathrm{Cov}(Y) = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2 & 2 & \cdots & 2 \\ 1 & 2 & 3 & 3 & \cdots & 3 \\ 1 & 2 & 3 & 4 & \cdots & 4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & 3 & 4 & \cdots & 10 \end{bmatrix}.$$

Do an example to show that this generalized least squares estimate give the sample mean $\bar{w}_{\cdot} = y_{10}/10$.

```
# 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
```

2.8 Variance-Bias Tradeoff

No new computing?

Chapter 3
Nonparametric Regression I

3.1 Simple Liner Regression

Nothing new here.

3.2 Polynomial regression

One way to 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 on simple linear regression, 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*.

3.2.1 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)
```

I'm not sure this next group of commands couldn't be replaced by `anova(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)
```

3.2.1.1 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)`.

3.2.2 Exploring the chosen polynomial

These are just the standard regression commands 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)
```

3.3 Overfitting Polynomial Regression

There are no new computing skills involved in this section.

3.4 Additional Spanning 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

3.4.1 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)
```

3.4.2 *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)
```

3.5 Partitioning

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 for 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)
```

3.5.1 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.

3.6 Splines

You would probably not want to do an entire spline fit without specialized software such as `library(splines)`. 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)

..48:70931 2.25296 ..21:620 0.000
x 0.35571 0.01208 29.447 0.000
(x..191)+ 0:13147


#Partition
xx=seq(180.5,211,.2)
yy1=-48.70931+0.35571*xx
yy2=-48.70931+.35571*xx+.13147*(xx-191)
plot(Temp,Pres,xlim=c(180.5,211),type="p")
#I have to fudge the lines so they meet visually
lines(xx[xx<=191.25],yy1[xx<=191.25],type="l")
lines(xx[xx>=191],yy2[xx>=191],type="l")
```

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))
```

3.7 Fisher's Lack-of-Fit 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.

3.8 Additive Effects Versus Interaction

No new computing.

3.10 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 (3.10.2)

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

To fit the polynomial version of the book's model (3.10.3) when $R = S$

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

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

Chapter 4
Alternative Estimates I

4.1 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.

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

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

4.2 Classical Ridge Regression

See Section 8.2.

One might also look at `lm.ridge` in Venable and Ripley's `MASS` package.

4.3 Lasso Regression

```
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 2.

```
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
```

or `Xs=scale( x1+x2+x3+x4+x5)` or

```
X=matrix(c(x1,x2,x3,x4,x5,ncol=5)
Xs = scale(X)
Xs
```

Or if you want, you could only center the variables or only rescale them without centering them by modifying
`scale(coleman[,2:6], center = TRUE, scale = TRUE)`

4.3.0.1 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","a5","x10","y"))
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,
tib
```

4.3.1 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 CV value
abline(minPLUSoneSE,0,lty=2) #looks like \lambda=0.45 is within the one SE from the minimum
# 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
```

4.4 Robust Estimation and Alternative Distances

For doing $\mathbf{L}^1$ regression you can use the program `l1fit` from the package `L1pack`. Another option might be `rq(y   model, tau=0.5))` in package `quantreg` seems to do it.

Program `rlm` from the `MASS` package will use Tukey's bi-weight loss function as well as those attributed to Huber and Hampel. `rlm(y   model, psi = psi.bisquare, init = "lts")`

Chapter 5
Variable Selection

5.1 Best Subset Selection

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.

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
```

You can also do this for logistic regression by setting up the inputs correctly, see (Christensen 1997, Section 4.4). Subsection 20.6.1 of the R code for ANREG-II illustrates the method. In my opinion this is a clearly better method than ones based on score tests.

5.2 Stepwise Variable Selection

R's `step` command is useful in many places, not just with `lm`. It chooses models based on the AIC criterion. *Plane Answers-V* contains a discussion of AIC. Another 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"`.

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

5.3 Variable Selection and Case Deletion

Nothing new here.

5.4 Discussion

5.5 Modern Forward Selection: Boosting, Bagging, and Random Forests

This is a simulation program I wrote before deciding that I could just discuss the three observation example to make clear the pluses and minuses of bagging. The simulation is also for one-sample problems and I believe is consistent with the results expressed in the example. For heavy tailed distributions, the median should work best, followed by (in order) the bagged median, the sample mean, the bagged midrange, and the midrange. For thin tailed distributions the midrange should be the best followed by the bagged midrange, the sample mean, the bagged median, and the median. For normal distributions (and many others) the sample mean should be the best.

For the Laplace distribution (and many other slightly unusual distributions) the package of repeated measures utilities `rmutil` is handy.

We generate `n` training observations and `ntst` test observations. Want to estimate the expected value of the distribution. Compute the sample mean, sample median, and sample midrange and compute the average of the bootstrap means, medians, and midranges as out estimates. Mean is optimal for normal and nonparametrically.

For normal data, sample mean is optimal and bootstrap cannot improve. For uniform data, midrange is optimal and bootstrap cannot improve. For laplace data, median is optimal and bootstrap cannot improve. (Nearly true for t(3).) For nonparametric data, sample mean is optimal. But bootstrap can improve the suboptimal estimates by averaging them.

The sample mean and the bootstrapped sample mean should be almost identical. See how close bootstrap is to optimal and how much better bootstrap is than suboptimal estimates.

This first program is for comparing estimates on a single data set. The following program evaluates things of many sets of data.

```
rm(list = ls())

#Determine sample sizes for test and training data
nfull=200
ntst=100
n=nfull-ntst

#Generate random test and training data.
#Compute mean, median, and midrange
Tst=rnorm(ntst,0,1)
Trn=rnorm(n,0,1)
#Will want to change those distributions
#Long tailed distributions include t(df) and Laplace
#Short tailed distributions include uniform(a,b)
xbar=mean(Trn)
xtld=median(Trn)
mid=(max(Trn)+min(Trn))/2

#Set up variables to save results from bootstrap samples.
B=1000
Bxbar=seq(1,B)
Bxtld=Bxbar
Bmid=Bxbar

#B bootstrap samples from training data
#B point estimates
```

```
for(k in 1:B)
{
Temp=sample(Trn,n,replace=T)
Bxbar[k]=mean(Temp)
Bxtld[k]=median(Temp)
Bmid[k]=(max(Temp)+min(Temp))/2
}

#Mean predictive sums of squares error
#From both regular and bootstrapped estimates.
PredSS=seq(1:9)
PredSS[1]=sum((Tst- xbar)^2)/ntst
PredSS[2]=sum((Tst- xtld)^2)/ntst
PredSS[3]=sum((Tst- mid)^2)/ntst
PredSS[4]=sum((Tst- mean(Bxbar))^2)/ntst
PredSS[5]=sum((Tst- mean(Bxtld))^2)/ntst
PredSS[6]=sum((Tst- mean(Bmid))^2)/ntst
PredSS[7]=sum((Tst- 0)^2)/ntst
PredSS[8]=8
PredSS[9]=9
#PredSS=
matrix(PredSS,3,3)
```

The idea is to see that for normal data the sample mean does well (bootstrap mean is about the same as the mean), but they do reasonably well no matter what distribution you use. For short tailed distributions, the midrange does well, bootstrap of midrange less well, median poor, bootstrap median better. For long tailed distributions, the median does well, bootstrap of median less well, midrange poor, bootstrap midrange better.

This program repeats the previous program many (`SS`) times.

```
rm(list = ls())
# set size of full data, test data, and implicitly training data.
nfull=200
ntst=100
n=nfull-ntst
# Define simulation size and bootstrap sample size.
SS=3000 # No. of data sets generated.
B=1000  # No. of boot samples from data

#  Define sizes for vectors.
PredSS=seq(1:9)
APress=c(0,0,0,0,0,0,0,0,0)
Bxbar=seq(1,B)
Bxtld=Bxbar
Bmid=Bxbar
```

```
# Simulation
for(kk in 1:SS)
{
# Generate data
Tst=rt(ntst,3)
Trn=rt(n,3)
#install.packages("rmutil")
#library(rmutil)
#Tst=rlaplace(ntst,0,1)
#Trn=rlaplace(n,0,1)
#Compute estimates.
xbar=mean(Trn)
xtld=median(Trn)
mid=(max(Trn)+min(Trn))/2

# Obtain estimates from bootstrapping
for(k in 1:B)
{
Temp=sample(Trn,n,replace=T)
Bxbar[k]=mean(Temp)
Bxtld[k]=median(Temp)
Bmid[k]=(max(Temp)+min(Temp))/2
}
# Prediction error variance for each estimate.
PredSS[1]=sum((Tst- xbar)^2)/ntst
PredSS[2]=sum((Tst- xtld)^2)/ntst
PredSS[3]=sum((Tst- mid)^2)/ntst
PredSS[4]=sum((Tst- mean(Bxbar))^2)/ntst
PredSS[5]=sum((Tst- mean(Bxtld))^2)/ntst
PredSS[6]=sum((Tst- mean(Bmid))^2)/ntst
PredSS[7]=sum((Tst- 0)^2)/ntst
APress=APress+PredSS
}
APress=APress/SS
matrix(APress,3,3)
```

5.5.1 Extreme Gradiant Boosting

I haven't explored this yet.

```
# check whether need install "drat"
#install.packages("egboost")
```

```
library(egboost)
bst <- xgboost(data = x, label = y, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,
               objective = "binary:logistic")
# the software appears to prefer input like
# data = train$data, label = train$label,
# objective = "binary:logistic" or "reg:linear"
# I think from CART, Class and Reg Trees
```

To use options from following program may need to use params=list().
This program is more advanced and specifies how `xgboost` “trains.”

```
xgb.train(params = list(), data, nrounds, watchlist = list(), obj = NULL,
    feval = NULL, verbose = 1, print.every.n = 1L,
    early.stop.round = NULL, maximize = NULL, ...)
booster = gbtree (default) or gblinear
#parameters for gblinear are for lasso/ridge regularization
#parameters for gbtree are for trees
#objective specifies learning task
objective = reg:linear, reg:logistic, binary:logistic
[no idea how logistics differ but binary seems more common],
multi:softmax (see NN appendix)
```

Chapter 6
Multiple Comparisons

These methods are all easy to apply. I'm looking around for software that applies them quite generally. **So far, these commands look like they are only for one-way ANOVA.** In *SL* one-way ANOVA is discussed in Chapter 2, alluded to relative to Fisher's lack-of-fit test in Chapter 3, is the first model addressed in Appendix B and also mentioned in Appendix C.

Throughout the chapter we assume that the following commands have been run. I think this is Mandel's data with the various factors other than C2 being recodings of treatment groups.

```
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 *ANREG-II* Section 12.4.

6.1 Bonferroni Corrections

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

6.2 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, main = NULL)
```

6.3 Least Significant Differences

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)`.

?? library(stats) ??

Chapter 7
Nonparametric Regression II

7.1 Linear Approximations

7.2 Simple Nonparametric Regression

7.3 Estimation

The big new computing issue with linear approximation nonparametric regression is creating the model matrix. There is a good chance that you could find R packages to help with these tasks. Frankly, it would be better if you programmed this stuff yourself. As I used to argue with balanced ANOVA problems, you should stop doing them on a hand calculator because it is is boring, not because you find it difficult. Doing it the hard way helps you learn what you are doing. The time to start doing it the easy way is when you know what you are doing.

7.3.1 Polynomials

We fit the polynomial model necessary for obtaining Figure 1.1.

```
rm(list = ls())
battery <- read.table(
        "C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

xb=mean(x)
xc=x-xb
```

```
poly = lm(y ~ xc+I(xc^2)+I(xc^3)+I(xc^4))
polys = summary(poly)
polys
anova(poly)

xx=seq(0,1,.01)
yy1= (14.5804 + 7.673 *(xx-.5) - 63.424 *(xx-.5)^2 -
25.737 *(xx-.5)^3 + 166.418 *(xx-.5)^4)

plot(x,y,type="p")
lines(xx,yy1,type="l")
```

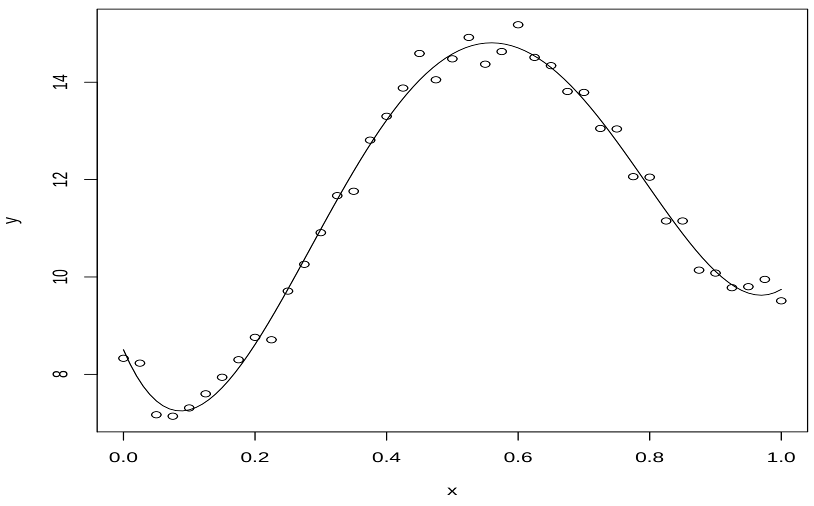


Fig. 7.1 Cosine model

7.3.2 Cosines

This produces Figures 7.2 and 7.3 by picking "p" appropriately. Currently has $p = 6$.

```
rm(list = ls())
battery <- read.table(
        "C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery


nf=length(x)
p=6
Phi=matrix(seq(1:(nf*p)),nf)

for(k in 1:p)
{
Phi[,k]=cos(pi*k*x)
#S=sin(pi*k*x)
}
#Phi

cos = lm(y~Phi)
coss = summary(cos)
coss
anova(cos)
Bhat=coefficients(cos)


xx=seq(0,1,.01)
nff=length(xx)
Phinew=matrix(seq(1:(nff*(p+1))),nff)
#Phinew
for(k in 1:(p+1))
{
Phinew[,k]=cos(pi*(k-1)*xx)
#S=sin(pi*k*xx)
}
Phinew

yy1=Phinew%*%Bhat


plot(x,y,type="p")
lines(xx,yy1,type="l")
par(mfrow=c(1,1))

# FOR SINES AND COSINES, CODE HAS NOT BEEN RUN
Phi=matrix{seq(1:nf*2*p),p*2}
for(k in 1:p)
{r=k
```

```
Phi[2r]=cos(pi*k*x)
Phi[2r+1]=sin(pi*k*x)
}
```

7.3.3 Haar wavelets

```
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

nf=length(x)
# p determines the highest level of wavelets used, i.e., m_{pk}
p=5
pp=nf*(2^p-1)
Phi=matrix(seq(1:pp),nf)
#Phi is the Haar wavelet model matrix WITHOUT a column of 1s


Phi[,1]=((as.numeric(x<=.5)-as.numeric(x>.5))
 *as.numeric(x<=1)*as.numeric(x>0)) #m_{1}
Phi[,2]=((as.numeric(x<=.25)-as.numeric(x>.25))
 *as.numeric(x<=.5)*as.numeric(x>0)) #m_{21}
Phi[,3]=((as.numeric(x<=.75)-as.numeric(x>.75))
 *as.numeric(x<=1)*as.numeric(x>.5)) #m_{22}

# this defines the rest
for(f in 3:p){
for(k in 1:(2^(f-1)))
{
a=(2*k-2)/2^f
b=(2*k-1)/2^f
c=2*k/2^f
kk=2^(f-1)-1+k
Phi[,kk]=((as.numeric(x<=b)-as.numeric(x>b))
*as.numeric(x<=c)*as.numeric(x>a))
}}
Phi
```

```
Haar = lm(y~Phi)
Haars = summary(Haar)
Haars
anova(Haar)
Bhat=coef(Haar)

# Plotting the fitted values for Haar wavelets.
xx=seq(0,1,.01)
nff=length(xx)
ppf=nff*(2^p)
Phinew=matrix(seq(1:ppf),nff)

Phinew[,1]=1
Phinew[,2]=((as.numeric(xx<=.5)-as.numeric(xx>.5))
 *as.numeric(xx<=1)*as.numeric(xx>0))
Phinew[,3]=((as.numeric(xx<=.25)-as.numeric(xx>.25))
 *as.numeric(xx<=.5)*as.numeric(xx>0))
Phinew[,4]=((as.numeric(xx<=.75)-as.numeric(xx>.75))
 *as.numeric(xx<=1)*as.numeric(xx>.5))

for(f in 3:p){
for(k in 1:(2^(f-1)))
{
a=(2*k-2)/2^f
b=(2*k-1)/2^f
c=2*k/2^f
kk=2^(f-1)+k
Phinew[,kk]=((as.numeric(xx<=b)-as.numeric(xx>b))
*as.numeric(xx<=c)*as.numeric(xx>a))
}}
Phinew



yy1=Phinew%*%Bhat

plot(x,y,type="p")
lines(xx,yy1,type="l")
```

7.3.4 Cubic Splines

For constructing splines from scratch, the R-code for *ANREG-II* has a section on splines. This is more systematic. There is another section later that discusses another option.

```
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery


nf=length(x)
#nk is the number of interior knots
nk=30  #  or 4
# Create a matrix the correct size for the predictor variables.
Phi=matrix(seq(1:(nf*(nk+3))),nf)
Phi
# Create matrix of predictor variables.
Phi[,1]=x
Phi[,2]=x^2
Phi[,3]=x^3

knot=seq(1:nk)/(nk+1)
knot
for(k in 1:nk)
{
Phi[,(k+3)]=(as.numeric(x>knot[k])*(x-knot[k]))^3
}
Phi  #Phi differs from book.  It has no column of 1s.

spln = lm(y~Phi)
splns = summary(spln)
splns
anova(spln)
Bhat=coefficients(spln)

xx=seq(0,1,.01)
nff=length(xx)
Phinew=matrix(seq(1:(nff*(nk+4))),nff)
Phinew[,1]=1
Phinew[,2]=xx
```

```
Phinew[,3]=xx^2
Phinew[,4]=xx^3
for(k in 1:nk)
{kk=k+4
Phinew[,kk]=(as.numeric(xx>knot[k])*(xx-knot[k]))^3
}
Phinew
yy1=Phinew%*%Bhat
plot(x,y,type="p")
lines(xx,yy1,type="l")
```

7.4 Variable Selection

Just get the sequential sums of squares printed from `lm`

7.5 Approximating-Functions with Small Support

7.5.1 Polynomial Splines

We already illustrated how to construct splines from scratch. The following is an example that illustrates computationally that polynomial splines agree with properly constructed bsplines.

7.5.1.1 Regular spline model

```
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery


nf=length(x)
#nk =  number of interior knots
nk=40  #  or 4
# d=dimension of poly
d=3
```

```
d=as.integer(d)
Phi=matrix(seq(1:(nf*(nk+d))),nf)
Phi
for(kk in 1:d)
{
Phi[,kk]=x^kk
}


knot=seq(1:nk)/(nk+1)
knot
for(k in 1:nk)
{
Phi[,(k+d)]=(as.numeric(x>knot[k])*(x-knot[k]))^d
}
Phi

spln = lm(y~Phi)
splns = summary(spln)
splns
anova(spln)
Bhat=coefficients(spln)

xx=seq(0,1,.01)
nff=length(xx)
Phinew=matrix(seq(1:(nff*(nk+d+1))),nff)
for(k in 1:(d+1))
{kk=k-1
Phinew[,k]=xx^kk
}
for(k in 1:nk)
{kk=k+d+1
Phinew[,kk]=(as.numeric(xx>knot[k])*(xx-knot[k]))^d
}
Phinew
yy1=Phinew%*%Bhat
plot(x,y,type="p")
lines(xx,yy1,type="l")
```

7.5.1.2 B-splines

B-splines for $d = 2,3$. Currently set for $d = 3$. For $d = 2$ change `Phi[,k]=Psi3` to `Phi[,k]=Psi2`.

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

```
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery


nf=length(x)
#nk = m = number of interior knots +1
nk=31  #  or 4
# d=dimension of poly
d=3
d=as.integer(d)
Phi=matrix(seq(1:(nf*(nk+d))),nf)


for(k in 1:(nk+d))
{
j=k-1
u=nk*x-j+d
# d=2 mother
Psi2 = (as.numeric(u >= 0))*(as.numeric(u <= 1))*(u^2/2)-
(as.numeric(u > 1))*(as.numeric(u <= 2))*((u-1.5)^2-.75)+
(as.numeric(u>2))*(as.numeric(u<=3))*((3-u)^2/2)
# d=3 mother
Psi3 = (as.numeric(u >= 0))*(as.numeric(u <= 1))*(u^3/3)+
(as.numeric(u > 1))*(as.numeric(u <= 2))*(-u^3+4*u^2-4*u+4/3)-
((as.numeric(u>2))*(as.numeric(u<=3))
  *(-(4-u)^3+4*(4-u)^2-4*(4-u)+4/3))+
(as.numeric(u>3))*(as.numeric(u<=4))*((4-u)^3/3)
Phi[,k]=Psi3                    #currently using d=3
}

bspln = lm(y~Phi-1)
bsplns = summary(bspln)
bsplns
anova(bspln)
Bhat=coefficients(bspln)

#setup for plotting fitted curve
xx=seq(0,1,.01)  #.01 determines grid
nff=length(xx)
#define matrix dimensions
Phinew=matrix(seq(1:(nff*(nk+d))),nff)
```

```
for(k in 1:(nk+d))
{
j=k-1
u=nk*xx-j+d
Psi2 = (as.numeric(u >= 0))*(as.numeric(u <= 1))*(u^2/2)+
(as.numeric(u > 1))*(as.numeric(u <= 2))*((u-1.5)^2-.75)+
(as.numeric(u>2))*(as.numeric(u<=3))*((3-u)^2/2)
Psi3 = (as.numeric(u >= 0))*(as.numeric(u <= 1))*(u^3/3)+
(as.numeric(u > 1))*(as.numeric(u <= 2))*(-u^3+4*u^2-4*u+4/3)+
((as.numeric(u>2))*(as.numeric(u<=3))
  *(-(4-u)^3+4*(4-u)^2-4*(4-u)+4/3))+
(as.numeric(u>3))*(as.numeric(u<=4))*((4-u)^3/3)
Phinew[,k]=Psi3
}
yy1=Phinew%*%Bhat
plot(x,y,type="p")
lines(xx,yy1,type="l")
```

Plot of Ψ_2

```
xx=seq(0,1,.01)  #.01 determines grid
u=xx*4-.5
Psi2 = (as.numeric(u >= 0))*(as.numeric(u <= 1))*(u^2/2)-
(as.numeric(u > 1))*(as.numeric(u <= 2))*((u-1.5)^2-.75)+
(as.numeric(u>2))*(as.numeric(u<=3))*((3-u)^2/2)
plot(u,Psi2,type="l")
```

Plot of Ψ_3

```
xx=seq(0,1,.01)  #.01 determines grid
u=xx*5-.5
Psi3 = (as.numeric(u >= 0))*(as.numeric(u <= 1))*(u^3/3)+
(as.numeric(u > 1))*(as.numeric(u <= 2))*(-u^3+4*u^2-4*u+4/3)+
((as.numeric(u>2))*(as.numeric(u<=3))
   *(-(4-u)^3+4*(4-u)^2-4*(4-u)+4/3))+
(as.numeric(u>3))*(as.numeric(u<=4))*((4-u)^3/3)
plot(u,Psi3,type="l")
```

7.5.1.3 More on B-splines

I have not yet checked out any of this.

Apparently a b-spline basis matrix for cubic splines can also be obtained using `ns(x, df = NULL, knots = NULL, intercept = FALSE, Boundary.knots = range(x))`. Although ns stands for natural spline, it apparently uses b-splines.

Also, you can do things other than cubic b-splines using `bs(x, df = NULL, knots = NULL, degree = 3, intercept = FALSE, Boundary.knots = range(x))`. Apparently, these are based on the command used below, `splineDesign`, which requires the package `splines`.

Use this to try and reconstruct results of Subsection 3.4.

Bsplines were also discussed Christensen et al. (2010) (*BIDA*) and the following code was written for their example. For the data in *BIDA*, 12 is much better although the fit is too wiggly in the first section and misses the point of inflection. We now create the B-spline basis. You need to have three additional knots at the start and end to get the right basis. We have chosen to the knot locations to put more in regions of greater curvature. We have used 12 basis functions for comparability to the orthogonal polynomial fit. (I think this is from Tim Hanson.)

```
# At some point you need to have run
#install.packages("splines")
library(splines)
knots <- c(0,0,0,0,0.2,0.4,0.5,0.6,0.7,0.8,0.85,0.9,1,1,1,1)
bx <- splineDesign(knots,x)
gs <- lm(y ~ bx)
matplot(x,bx,type="l",main="B-spline basis functions")
matplot(x,cbind(y,gs$fit),type="pl",ylab="y",pch=18,
    lty=1,main="Spline fit")
# The basis functions themselves are shown
> gs <- lm(y ~ bx -1)
```

The package `splines2` supposedly constructs b-spline model matrices.

7.5.2 Fitting Local Functions

7.5.3 Local regression

The default `loess` fit seems to oversmooth the data. It gives $R^2 = 0.962$.

```
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

lr = loess(y ~ x)
summary(lr)
xx=seq(0,1,.01)  #.01 determines grid
```

```
lr2=predict(lr, data.frame(x = xx))
plot(x,y)
lines(xx,lr2,type="l")
(cor(y,lr$fit))^2
```

The package `np` does *kernel smoothing*.

7.6 Nonparametric Multiple Regression

The package `gam` fits generalized additive models using smoothing splines or local regression

7.6.1 Redefining ϕ and the Curse of Dimensionality

7.6.2 Reproducing Kernel Hilbert Space Regression

EXAMPLE 7.6.1. I fitted the battery data with the R language's `lm` command using the three functions $R(u,v) = (u'v)^4$, $R(u,v) = (1+u'v)^4$, and $R(u,v) = 5(7+u'v)^4$. I got identical fitted values $\hat{y}_i$ to those from fitting a fourth degree polynomial.

```
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

# obtain the X data matrix from the untransformed data
lin=lm(y ~ x)
X <- model.matrix(lin)



# fit the 4th degree polynomial regression 4 equivalent ways
xb=mean(x)
xc=x-xb
poly = lm(y ~ xc+I(xc^2)+I(xc^3)+I(xc^4))
polys = summary(poly)
polys
anova(poly)
```

```
Phi <- model.matrix(poly)
Z=Phi[,-1]
poly2 = lm(y ~ Z)
poly3 = lm(y ~ Phi - 1)
poly4 = lm(y ~ (Phi%*%t(Phi))-1)


#create Rtilde
# Three options for R available
# Comment out the two you don't want
d=4
nf=length(x)
Rtilde=matrix(seq(1:(nf*nf)),nf)

for(k in 1:nf)
{
u=X[k,]
for(kk in 1:nf)
{
v=X[kk,]
Rtilde[k,kk]=(1+t(u)%*%v)**d   # polynomial
#Rtilde[k,kk]=exp(-1000*t(u-v)%*%(u-v)) # Gaussian
#Rtilde[k,kk]=tanh(1* (t(u)%*%v) +0)   # hyperbolic tangent
}
}
# Minor adjustments to the above
# needed to incorporate constants

# Fit the RKHS regression and compare fitted values
poly5 = lm(y ~ Rtilde-1)
# or lm(y ~ Rtilde)
poly5$fit
poly$fit
poly2$fit
poly3$fit
poly4$fit

# Check if Rtilde is nonsingular
eigen(Rtilde)$values
Rtilde%*%solve(Rtilde)



# Fit hyperbolic tangent
# Rerun earlier code for
# appropriate Rtilde
tanhh = lm(y ~ Rtilde-1)
summary(tanhh)

# Plot results
xx=seq(0,1,.01)
yy1=tanhh$coef[1] * tanh(1*(1+xx*x[1])+0)+
tanhh$coef[2] * tanh(1*(1+xx*x[2]) +0)+
tanhh$coef[3] * tanh(1*(1+xx*x[3])+0) +
tanhh$coef[4] * tanh(1*(1+xx*x[4])+0)+
tanhh$coef[11]*tanh(1*(1+xx*x[11])+0) +
tanhh$coef[16]*tanh(1*(1+xx*x[16])+0) +
tanhh$coef[29]*tanh(1*(1+xx*x[29])+0) +
tanhh$coef[41] * tanh(1*(1+xx*x[41])+0)
plot(x,y,type="p")
lines(xx,yy1,type="l")
```

See Chapter 3.

7.7 Testing Lack of Fit in Linear Models

Not much new computing here.

7.8 Regression Trees

Also known as *recursive partitioning*

Package and program `rpart`.

```
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

#install.packages("rpart")
library(rpart)
fit=rpart(y~x3+x5,method="anova",control=rpart.control(minsplit=7))
# minsplit=7 means that if a partition set contains less than 7 observations
# a split will not be attempted
# I set it at 7 so as to get a split based on x5
# The default minimum number of observations in a partition set
# is minsplit/3 rounded off.
# The default minsplit is 20, so not very interesting for data with n=20.


printcp(fit)
plotcp(fit)
rsq.rpart(fit)
print(fit)
summary(fit)
plot(fit)
text(fit)
post(fit, file="C:\\E-drive\\Books\\ANREG2\\newdata\\colemantree.pdf")  cr

par(mfrow=c(3,2))
plot(x3,x5,main="Recursive Partitioning")
lines(c(-11.35,-11.35),c(5,8),type="o",lty=1,lwd=1)
plot(x3,x5,main="Recursive Partitioning")
lines(c(-11.35,-11.35),c(5,8),type="o",lty=1,lwd=1)
lines(c(8.525,8.525),c(5,8),type="o",lty=1,lwd=1)
plot(x3,x5,main="Recursive Partitioning")
lines(c(-11.35,-11.35),c(5,8),type="o",lty=1,lwd=1)
lines(c(8.525,8.525),c(5,8),type="o",lty=1,lwd=1)
lines(c(6.235,6.235),c(5,8),type="o",lty=1,lwd=1)
plot(x3,x5,main="Recursive Partitioning")
lines(c(-11.35,-11.35),c(5,8),type="o",lty=1,lwd=1)
lines(c(8.525,8.525),c(5,8),type="o",lty=1,lwd=1)
lines(c(12.51,12.51),c(5,8),type="o",lty=1,lwd=1)
lines(c(6.235,6.235),c(5,8),type="o",lty=1,lwd=1)
plot(x3,x5,main="Recursive Partitioning")
lines(c(-11.35,-11.35),c(5,8),type="o",lty=1,lwd=1)
lines(c(8.525,8.525),c(5,8),type="o",lty=1,lwd=1)
lines(c(12.51,12.51),c(5,8),type="o",lty=1,lwd=1)
lines(c(6.235,6.235),c(5,8),type="o",lty=1,lwd=1)
lines(c(-11.35,6.235),c(5.675,5.675),type="l",lty=1,lwd=1)
plot(x3,x5,main="Alternative Second Partition")
lines(c(-11.35,-11.35),c(5,8),type="o",lty=1,lwd=1)
lines(c(-11.35,20),c(6.46,6.46),type="l",lty=1,lwd=1)
par(mfrow=c(1,1))




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

```
anova(co)
# The default for anova gives 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")
```

Do a regression tree on ANOVA data. Cookie Moisture

Regression trees don't seem to be very good without "bagging" them into random forests.

http://cran.r-project.org/web/packages/randomForest/index.html. http://cran.r-project.org/web/packages/ranger/index.html

Useful commands? natural splines `ns`, `modcv()`

7.9 Regression on Functional Predictors

Chapter 8
Alternative Estimates II

8.1 Introduction

libraries
  `lasso2` program `l1ce`
  `glmnet` library and program

8.2 Ridge Regression

Below are programs for fitting the augmented linear model. One might also look at `lm.ridge` in Venable and Ripley's `MASS` package.

```
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

nf=length(x)
p=10
Phi=matrix(seq(1:(nf*(p+1))),nf)

for(k in 1:(p+1))
{
Phi[,k]=cos(pi*(k-1)*x)
#S=sin(pi*k*x)
}
```

```
#This section fits the p-1=6 model.
Phi6=Phi[,c(2,3,4,5,6,7)]

cos6 = lm(y~Phi6)
coss6 = summary(cos6)
coss6
anova(cos6)
Bhat6=coefficients(cos6)
Bhat6=c(Bhat6,0,0,0,0)
Bhat6

Create and fit the augmented Ridge model

w=seq(1:10)
zip=w-w
w=w
w=w*w
Dw=.2*diag(w)
Dwx=cbind(zip,Dw)
Dwx
XR=rbind(Phi,Dwx)
XR
YR=c(y,zip)
cosr=lm(YR ~ XR-1)
BhatR=coefficients(cosr)
BhatR

(cor(y,Phi%*%Bhat6)^2)
(cor(y,Phi%*%BhatR)^2)

xx=seq(0,1,.01)
nff=length(xx)
Phinew=matrix(seq(1:(nff*(p+1))),nff)
for(k in 1:11)
{
Phinew[,k]=cos(pi*(k-1)*xx)
#S=sin(pi*k*xx)
}
Phinew

yy1=Phinew%*%Bhat6
yyr=Phinew%*%BhatR

plot(x,y,type="p")
lines(xx,yy1,type="l",lty=5)
lines(xx,yyr,type="l")
par(mfrow=c(1,1))
```

It is most natural to apply ridge regression (and the lasso) 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.

```
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
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)
```

8.3 Lasso Regression

You can get a lot information on lasso from Rob Tibshirani's website: http://statweb.stanford.edu/~tibs/lasso.html. There is another example in the R code for Section 10.2 of *ANREG-II*. This is for comparing the $p = 30$ cosine lasso fit with with the $p = 6$ cosine least squares fit.

```
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

nf=length(x)
p=30
```

```
Phi=matrix(seq(1:(nf*(p+1))),nf)

for(k in 1:(p+1))
{
Phi[,k]=cos(pi*(k-1)*x)
#S=sin(pi*k*x)
}

#This section fits the p-1=6 model.
Phi6=Phi[,c(2,3,4,5,6,7)]

cos6 = lm(y~Phi6)
coss6 = summary(cos6)
coss6
anova(cos6)
Bhat6=coefficients(cos6)
Bhat6=c(Bhat6,rep(0,24))
Bhat6


cos = lm(y~Phi-1)
coss = summary(cos)
coss
anova(cos)
Bhat=coefficients(cos)


# install.packages("lasso2")
library(lasso2)
tib <- l1ce(y ~ Phi[,-1],data=battery,standardize=FALSE,bound=.5)
# This is the default boundary.
# Generalized lasso requires weights=
BhatL=coef(tib)
#tib <- l1ce(y ~ Phi[,-1],bound = 0.5+(seq(1:10)/100))
tib
BhatL

(cor(y,Phi%*%Bhat6)^2)
(cor(y,Phi%*%Bhat)^2)
(cor(y,Phi%*%BhatL)^2)

xx=seq(0,1,.01)
nff=length(xx)
Phinew=matrix(seq(1:(nff*(p+1))),nff)
#Phinew
```

```
for(k in 1:(p+1))
{
Phinew[,k]=cos(pi*(k-1)*xx)
#S=sin(pi*k*xx)
}
#Phinew
yy6=Phinew%*%Bhat6
yyL=Phinew%*%BhatL
yy30=Phinew%*%Bhat

plot(x,y,type="p")
lines(xx,yy6,type="l",lty=5)
lines(xx,yy30,type="l",lty=4)
lines(xx,yyL,type="l")
par(mfrow=c(1,1))
```

8.4 Geometry

```
#install.packages("ellipse") #Do this only once on your computer
library(ellipse)
#SHRINKAGE, NO ZEROS

b1=1
b2=2
A = matrix(c(1,.9,.9,2),2,2, dimnames = list(NULL, c("b1", "b2")))
A
E <- ellipse(A, centre = c(b1, b2), t = .907, npoints = 100)
E1 <- ellipse(A, centre = c(b1, b2), t = .5, npoints = 100)
x=seq(0,1,.05)
y=1-x
y1=-1+x
x1=-x
y2=1+x1
y3=-1-x1
plot(E,type = ’l’,ylim=c(-1,3),xlim=c(-1,3),
     xlab=expression(~beta[1]),
 ylab=expression(~beta[2]),main=expression(~delta==1))
text((b1+.1),(b2-.15),expression(hat(~beta)),lwd=1,cex=1)
#plot(E,type = ’l’,ylim=c(-1,3),xlim=c(-1,3),
#xlab=expression(beta[1]),ylab=expression(beta[2]))
lines(E1,type="l",lty=1)
lines(x,y,type="l",lty=1)
lines(x,y1,type="l",lty=1)
lines(x1,y2,type="l",lty=1)
lines(x1,y3,type="l",lty=1)
lines(0,0,type="p",lty=3)
lines(b1,b2,type="p",lty=3)
#text((b1+.1),(b2-.15),"(b1,b2)",lwd=1,cex=1)
```

8.5 Two Other Penalty Functions

Chapter 9
Classification

See Chapter 20 of *ANREG-II* for a more extensive discussion of logistic regression and associated code. The ANREG-II discussion and code includes:

- Goodness of fit tests.
  Don't use the Hosmer-Lemeshow chi-squared test. Only use the Pearson and Likelihood ratio tests when you have binomial data with all the N_is reasonably large!
- Assessing predictive probabilities.
- Case diagnostics.
- Model testing.
- Multiple logistic regression.
- Best subset logistic regression.
- Stepwise logistic regression.
- ANOVA models.
- Ordered categories.

`glm` fits generalized linear models but it does not incorporate penalties except through augmenting the data.

Library `LiblineaR` has program `LiblineaR` that performs logistic regression and SVM with both $\mathbf{L}^1$ (lasso) and $\mathbf{L}^2$ (ridge) regularization (penalties).

9.1 Binomial Regression

9.1.1 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
```

9.1.2 Data Augmentation Ridge Regression

Go to last section of this chapter.

9.2 Binary Prediction

O-ring data analysis.

```
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
```

9.2.1 Figure Construction

In order to construct the figures, I had to fit the models. So the information on fitting the models is embedded in the code for the figures.

Figure 9.1 involves fitting logistic and probit regressions and an SVM classifier. The linear structures are $\beta_0 + \beta_1 TL + \beta_2 PL$, i.e., linear in TL and PL.

```
rm(list = ls())
cush <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1c.DAT",
  sep="",col.names=c("Type","Tetra","Preg"))
attach(cush)
cush
TL = log(Tetra)
PL = log(Preg)
Tp=Type-2
Tp2=3-Type
# Tp2 is 1 for Bilat hyp and 0 for Carcin

# Plot of points and discrimination curves
xx2=c( 8.3, 3.8, 3.9, 7.8, 9.1, 15.4, 7.7, 6.5, 5.7, 13.6)
yy2=c(1.00, .20, .60, 1.20, .60, 3.60, 1.60, .40, .40, 1.60)
xx3=c(10.2,  9.2,  9.6, 53.8, 15.8)
yy3=c(6.40, 7.90, 3.10, 2.50, 7.60)
```

```
x2=log(xx2)
x3=log(xx3)
y2=log(yy2)
y3=log(yy3)
plot(x2,y2, pch=16, ylab="log(Pregnanetriol)",
 ylim=c(-2,3),xlim=c(1,4.5),xlab="log(Tetrahydrocortisone)")
points(x3, y3, pch=22)
legend("bottomright",c("Bilateral Hyperplasia","Carcinoma"),
pch=c(16,22))


# LOGISTIC REGRESSION
ac2 = glm(Tp2 ~ TL + PL,family=binomial)
#summary(ac2)
#anova(ac2)
#post=c(Type,ac2$fit,1-ac2$fit)
#PropProb=
#  matrix(post,15,3,dimnames = list(NULL,c("Group","B","C")))
#PropProb

x1=seq(1,4.5,.01)
b=ac2$coef
y2=(b[1]+b[2]*x1)/-b[3]
lines(x1,y2,type="l")
legend("topright",c("Logistic","Probit","SVM"),lty=c(1,2,5))

# PROBIT REGRESSION
ac3=glm(Tp2 ~ TL + PL,family=binomial(link="probit"))
b=ac3$coef
y3=(b[1]+b[2]*x1)/-b[3]
lines(x1,y3,type="l",lty=2)


#  SVM
#install.packages("e1071")
library(e1071)
T=factor(Type)
#  Typically you would want to have
#   \texttt{scale=T} in \texttt{svm}.
fit <- svm(T ~ TL + PL,kernel="linear",scale=F)
fit$SV
fit$coefs
fit$rho
x1=seq(1,4.5,.005)
b=t(fit$SV)%*%fit$coefs
```

```
#solve <(x1,ySVM)'b>=fit$rho
ySVM=(-fit$rho+b[1]*x1)/-b[2] # = (fit$rho - b[1]*x1)/b[2]
lines(x1,ySVM,type="l",lty=5)
```

9.2.2 Generalized Linear Models

This produces Figures 9.2 and 9.4 that compares the logistic and probit loss functions and the logistic and SVM loss functions, respectively.

```
rm(list = ls())
par(mfrow=c(1,2))
x=seq(-3.5,3.5,.05)
# Logit
y=2*log((1+exp(-x)))
# SVM
##y2=(1-x)*as.numeric((1-x)>0)
# Probit
y2=-2*log(pnorm(x))
# This gives HALF of a SVM
#y2=-2*log(1-pexp(-.5*x+.5))
plot(x,y,type="l",xlab=expression(x^T~beta),ylab="Loss")
lines(x,y2,,lty=5)
#legend("topright",c("Logistic","SVM"),lty=c(1,5))
legend("topright",c("Logistic","Probit"),lty=c(1,5))
legend("bottomleft",c("y=1"))
# Logit
y=-2*log(1/(1+exp(x)))
# SVM
##y4=(x+1)*as.numeric((x+1)>0)
# Probit
y4=-2*log(1-pnorm(x))
# This does NOT give the other half SVM
#y4=-2*log(pexp(-.5*x+.5))
plot(x,y,type="l",xlab=expression(x^T~beta),ylab="Loss")
lines(x,y4,,lty=5)
#legend("topleft",c("Logistic","SVM"),lty=c(1,5))
legend("topleft",c("Logistic","Probit"),lty=c(1,5))
legend("bottomright",c("y=0"))
par(mfrow=c(1,1))
```

9.3 Binary Generalized Linear Model Estimation

9.4 Linear Prediction Rules

This 3-part program produces Figures 9.3? and 9.5?. They involves fitting logistic and probit regressions and an SVM classifier. The linear structures are quadratic in TL and PL. Part 1 plots the data. Part 2 plots the two regressions. Part 3 fits the default SVM classifier and has commented out the SVM classifier with the tuning parameter reduced by a factor of 100 (cost increased to 100).

9.4.0.1 Plotting the data

```
rm(list = ls())
cush <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1c.DAT",
  sep="",col.names=c("Type","Tetra","Preg"))
attach(cush)
cush
T=factor(Type)
TL = log(Tetra)
PL = log(Preg)
plot(TL[T==2], PL[T==2], pch=16, ylab="log(Pregnanetriol)",
 ylim=c(-2,3),xlim=c(1,4.5),xlab="log(Tetrahydrocortisone)")
points(TL[T==3], PL[T==3], pch=22)
legend("bottomright",c("Bilateral Hyperplasia","Carcinoma"),
         pch=c(16,22))
legend("topleft",c("Logistic","Probit","SVM"),lty=c(1,2,5))
```

9.4.0.2 Quadratic Logistic and Probit Regression

This is a continuation of the previous plotting program.

```
TL2=TL*TL
PL2=PL*PL
TPL=PL*TL
Tp=Type-2
Tp2=3-Type
# Tp2 is 1 for Bilat hyp and 0 for Carcin


ac2 = glm(Tp2 ~ TL + PL + TL2 + PL2 + TPL,
        family=binomial)
```

```
#solve quadratic equation
x1=seq(1,4.5,.1)
bb=ac2$coef
prior=.5
c=bb[1]+bb[2]*x1+bb[4]*x1^2
b= bb[3] + bb[6]*x1
a=bb[5]
delta=b^2-4*a*c
yLR = (-b+sqrt(delta))/(2*a)
yLR2 = (-b-sqrt(delta))/(2*a)
lines(x1,yLR,type="l",lty=1)
lines(x1,yLR2,type="l",lty=1)


ac3 = glm(Tp2 ~ TL + PL + TL2 + PL2 + TPL,
              family=binomial(link="probit"))


#solve quadratic equation
x1=seq(1,4.5,.1)
bb=ac3$coef
prior=.5
c=bb[1]+bb[2]*x1+bb[4]*x1^2
b= bb[3] + bb[6]*x1
a=bb[5]
delta=b^2-4*a*c
yPR = (-b+sqrt(delta))/(2*a)
yPR2 = (-b-sqrt(delta))/(2*a)
lines(x1,yPR,type="l",lty=1)
lines(x1,yPR2,type="l",lty=1)
```

9.4.0.3 Quadratic SVM

This is a continuation of the previous program. Typically you would want to have `scale=T` in `svm`, which is the default.

```
#install.packages("e1071")
library(e1071)
T=factor(Type)
# define SVM and display outputs
fit <- svm(T ~ TL + PL,kernel="polynomial",degree=2,
        gamma=1,coef0=1,scale=F)
# Next line reduces tuning parameter, i.e., increases cost.
#fit <- svm(T ~ TL + PL,kernel="polynomial",degree=2,
        gamma=1,coef0=1,scale=F,cost=100)
fit$SV
fit$coefs
fit$rho
fit$fitted # shows groups that each case was allocated to
# fitted shows that my parabolic solution is reasonable.
# predict(fit)

# define and solve quadratic equation.
# add curves to previous plot
# must solve matrix equation that involves inner products
x1=seq(1,4.5,.01)
w=fit$coefs
c=-fit$rho + sum(w)+2*sum(w*fit$SV[,1])*x1
        +sum(w*(fit$SV[,1])^2)*x1^2
b=2*sum(w*fit$SV[,2]) + 2*sum(w*fit$SV[,1]*fit$SV[,2])*x1
a=sum(w*(fit$SV[,2])^2)

delta=b^2-4*a*c
ySVM = (-b+sqrt(delta))/(2*a)
ySVM2 = (-b-sqrt(delta))/(2*a)
lines(x1,ySVM,type="l",lty=5)
lines(x1,ySVM2,type="l",lty=5)


# A plot that svm provides
plot(fit,cush)
```

We now illustrate how to get the SVM prediction results from output (in R, `fit = svm(...)`) that provides the matrix of $d-1$ dimensional support vectors X_S (`fit$SV`) which is $s \times (d-1)$ and extracted from X, the negative intercept K (`fit$rho`), and the coefficients w (`fit$coef`) that are the nonzero coefficients of $\frac{1}{2k}(\lambda'_{11}, -\lambda'_{10})$. The elements of the vector w are positive for one group and negative for the other. The two groups are identified as positive and negative. As in Chapter 3 of *ALM-III*, the kernel function (specified in the program) defines the inner product between two vectors. To predict the group of a new vector x, evaluate the matrix expression

$$\begin{bmatrix} < x, x_{S1} > \\ \vdots \\ < x, x_{Ss} > \end{bmatrix}' w - K.$$

The sign of result determines the group allocation. Note that this simplifies greatly when using the standard Euclidean inner product.

Exercise: Manually scale the variables, run `svm` with `scale=F` and compute $\hat{\beta}_*$ and $\hat{\beta}_0$ as discussed in the book. Now run `svm` with `scale=T` and compute

$\hat{\beta}_*$ and $\hat{\beta}_0$. How do you transform $\hat{\beta}_*$ and $\hat{\beta}_0$ computed with `scale=T` into an appropriate vector $\hat{\beta}$ on the original scale of the data?

9.5 Support Vector Machines

The computations were made in the previous section.
http://svms.org/tutorials/

9.6 A Simple Example Not in *ALM-III*

This example is not in the book. It was used for classroom discussion.

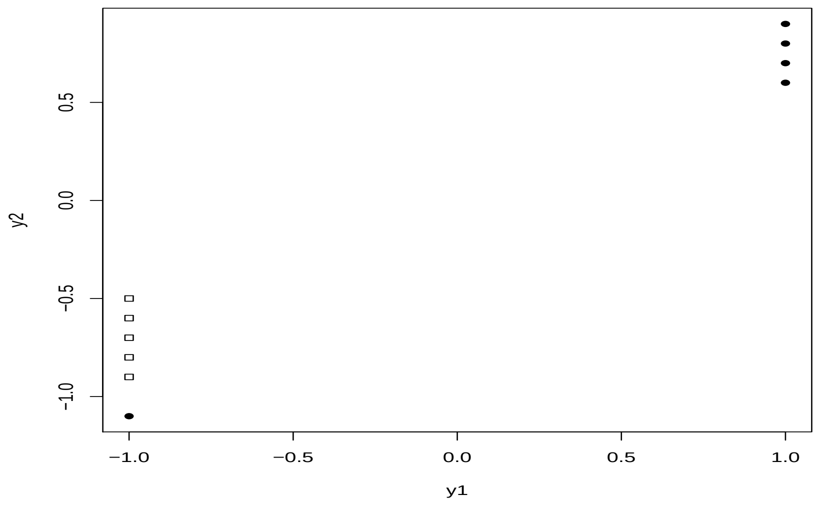


Fig. 9.1 Simple Binary Data.

```
rm(list = ls())
T=c(1,1,1,1,1,0,0,0,0,0)
TT=factor(T)
Y1=c(1,1,1,1,-1,-1,-1,-1,-1,-1)
Y2=c(.9,.8,.7,.6,-1.1,-.9,-.8,-.7,-.6,-.5)
Y11=Y1[T==1]
Y10=Y1[T==0]
```

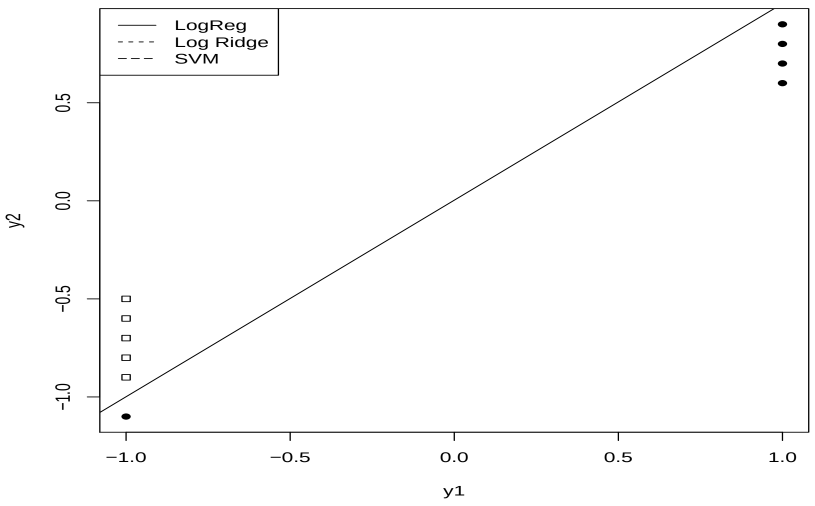


Fig. 9.2 Logistic Regression.

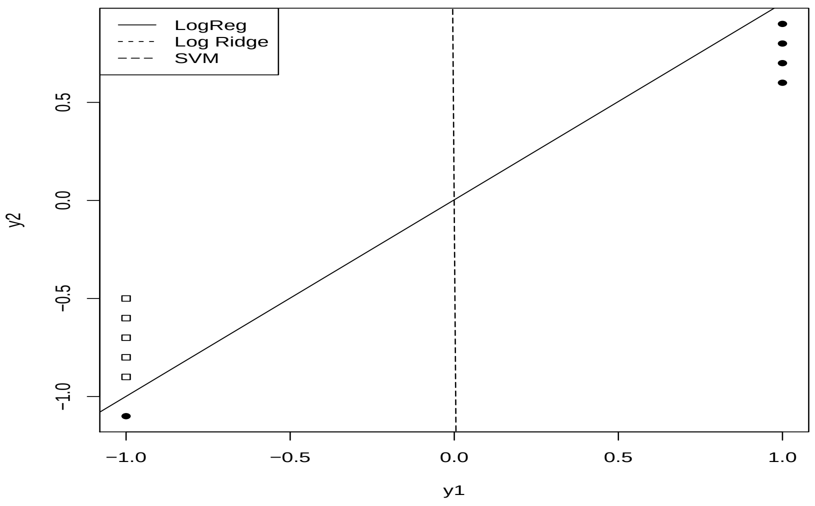


Fig. 9.3 Logistic Discrimination and R default SVM.

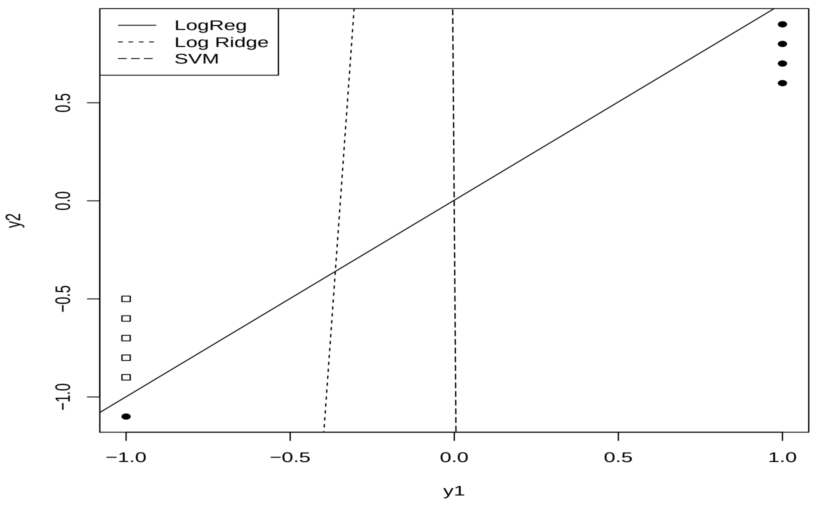


Fig. 9.4 Logistic Discrimination, R default SVM, LR augmented data ridge.

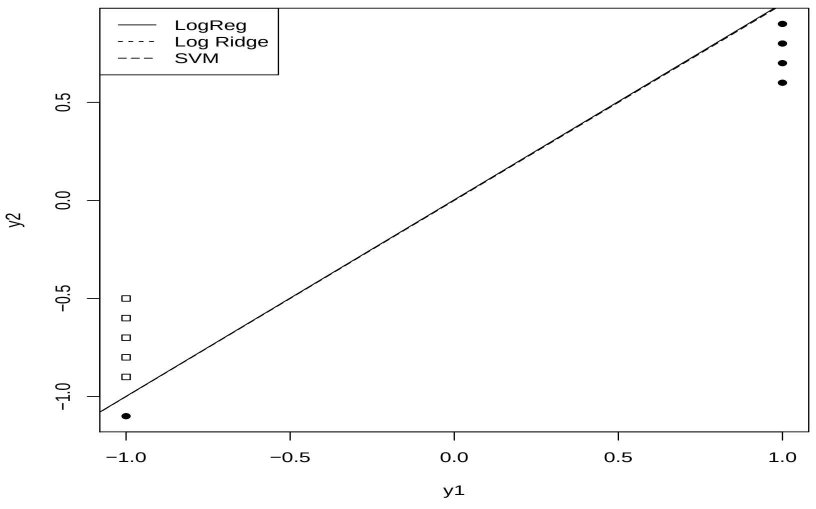


Fig. 9.5 Logistic Discrimination and SVM with C=100.

```
Y21=Y2[T==1]
Y20=Y2[T==0]
plot(Y11,Y21, pch=16, ylab="y2",xlab="y1")
 #ylim=c(-2,3),xlim=c(1,4.5),
points(Y10, Y20, pch=22)
x1=seq(-1.1,1.1,.05)
#lines(x1,x1)


ac2 = glm(T ~ Y1 + Y2,family=binomial)
prior=.5 #prior probability for first group (coded as 1)
n=length(T)
n1=5  #number of observations in first group
n2=n-n1

b=ac2$coef
yLR=(b[1]+log(n2/n1)+log(prior/(1-prior))+b[2]*x1)/-b[3]
lines(x1,yLR,type="l")
legend("topleft",c("LogReg","Log Ridge","SVM"),lty=c(1,2,5))



#install.packages("e1071")
library(e1071)
fit <- svm(TT ~ Y1 + Y2,kernel="linear",scale=F)
# add cost=100 and get LR.  Default is cost=1
fit$SV
fit$coefs
fit$rho
b=t(fit$SV)%*%fit$coefs
ySVM=(-fit$rho+b[1]*x1)/-b[2]
# = (fit$rho - b[1]*x1)/b[2]
lines(x1,ySVM,type="l",lty=5)


# No new software logistic ridge
k=1
TR=c(T,.5,.5)
Y1R =c(Y1,1,0)
Y2R =c(Y2,0,1)
W=c(Y1/Y1,k,k)
J=c(Y1/Y1,0,0)

acR = glm(TR ~ J + Y1R + Y2R -1,family=binomial,weights=W)
prior=.5 #prior probability for first group (coded as 1)
```

```
n=length(T)
n1=5  #number of observations in first group
n2=n-n1

b=acR$coef
yLRR=(b[1]+log(n2/n1)+log(prior/(1-prior))+b[2]*x1)/-b[3]
lines(x1,yLRR,type="l",lty=2)
```

Chapter 10
Discrimination and Allocation

An overview of R procedures for multivariate analysis is available at https://cran.r-project.org/web/views/Multivariate.html.

EXAMPLE 10.0.1. Read in the data and plot the data as follows.

```
rm(list = ls())
cush <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1.DAT",
  sep="",col.names=c("Type","Tetra","Preg"))
attach(cush)
cush
#summary(cush)
TL = log(Tetra)
PL = log(Preg)
```

The plot from ANREG-II

```
xx1=c(  3.1,  3.0,   1.9,  3.8,  4.1,  1.9)
yy1=c(11.70, 1.30,  0.10, 0.04, 1.10, 0.40)
xx2=c( 8.3,  3.8,  3.9,  7.8,  9.1, 15.4,  7.7,  6.5,  5.7, 13.6)
yy2=c(1.00, 0.20, 0.60, 1.20, 0.60, 3.60, 1.60, 0.40, 0.40, 1.60)
xx3=c(10.2,  9.2,  9.6, 53.8, 15.8)
yy3=c(6.40, 7.90, 3.10, 2.50, 7.60)
x1=log(xx1)
x2=log(xx2)
x3=log(xx3)
y1=log(yy1)
y2=log(yy2)
y3=log(yy3)
plot(x1,y1, pch=3, ylab="log(Pregnanetriol)",
 ylim=c(-4,4),xlim=c(0,5),xlab="log(Tetrahydrocortisone)")
points(x2, y2, pch=16)
points(x3, y3, pch=22)
legend("bottomright",c("Adenoma",
 "Bilateral Hyperplasia","Carcinoma"),pch=c(3,16,22))
```

10.1 The General Allocation Problem

10.1.1 Figure 2

```
rm(list = ls())
par(mfrow=c(2,1))
x=seq(-1,8,.1)
y=dnorm(x,2,1)
y1=dnorm(x,5,1)
plot(x,y,xlab="y",ylab="f(y)",type="l",lty=1,xlim=c(-1,8))
lines(x,y1,lty=1)
points( 3.5, 0, pch=16)

x=seq(-1,11,.01)
y=dnorm(x,2,1)
y1=dnorm(x,5,3)
plot(x,y,,ylab="f(y)",xlab="y",type="l",lty=1,xlim=c(-1,11))
lines(x,y1,lty=1)
c= 11
b= -26
a=8
delta=b^2-4*a*c
M1 = (-b+sqrt(delta))/(2*a)
M2 = (-b-sqrt(delta))/(2*a)
points( c(M1,M2),c(0,0), pch=16)
points( c(-.31,3.595),c(0,0), pch=22)
lines( c(-.31,-.31),c(0,.0277))
lines( c(3.595,3.595),c(0,.114))
legend("topright",c("QDA","Mahalanobis"),pch=c(22,16))
cbind(x,y,y1)
par(mfrow=c(1,1))
```

10.1.2 One dimensional plots

```
rm(list = ls())
par(mfrow=c(2,1))
x=seq(-1,8,.1)
y=dnorm(x,2,1)
y1=dnorm(x,5,1)
plot(x,y,type="l",lty=1,xlim=c(-1,8))
lines(x,y1,lty=1)
points( 3.5, 0, pch=16)

x=seq(-1,11,.01)
y=dnorm(x,2,1)
y1=dnorm(x,5,3)
plot(x,y,type="l",lty=1,xlim=c(-1,11))
lines(x,y1,lty=1)
c= 11
b= -26
a=8
delta=b^2-4*a*c
M1 = (-b+sqrt(delta))/(2*a)
M2 = (-b-sqrt(delta))/(2*a)
points( c(M1,M2),c(0,0), pch=16)
points( c(-.31,3.595),c(0,0), pch=22)
lines( c(-.31,-.31),c(0,.0277))
lines( c(3.595,3.595),c(0,.114))
legend("topright",c("QDA","Mahalanobis"),pch=c(22,16))
cbind(x,y,y1)
par(mfrow=c(1,1))
```

It is surprisingly futzy to transform from Cartesian to polar coordinates. I actually drew the plots by generating the data in polar coordinates and making the simpler transformation to Cartesian. The library `useful` has programs for doing both.

```
rm(list = ls())
n1=200
n2=200
yt11=runif(n1,1.5,1.75)
yt12=runif(n1,0,2*pi)
yt21=runif(n2,2.25,2.45)
yt22=runif(n2,0,2*pi)
y11=yt11*sin(yt12)
y12=yt11*cos(yt12)
y21=yt21*sin(yt22)
y22=yt21*cos(yt22)
```

```
plot(y21,y22,xlab=expression(y[1]),
 ylab=expression(y[2]),main="Discriminant Analysis")
lines(y11,y12,type="p",pch=4)

plot(yt11,yt22,xlim=c(1.5,2.5))
lines(yt21,yt22,type="p",pch=4)
```

In *ALM-III* the Examples are accumulated into Section 5 but here we give the linear the examples in sections that describe the theory. **Don't think this is still true.**

10.2 Estimated Allocation and QDA

The following commands reproduce Table 10.3 associated with Example 10.5.1.

```
rm(list = ls())
cush <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1.DAT",
  sep="",col.names=c("Type","Tetra","Preg"))
attach(cush)
cush
#summary(cush)
TL = log(Tetra)
PL = log(Preg)
T=factor(Type)

library(MASS)
fit <- qda(T ~ TL + PL,prior=c(1/3,1/3,1/3))
fit
pfit=predict(fit)
pfit
ct <- table(pfit$class,T)
ct

fit2 <- qda(T ~ TL + PL,prior=c(1/3,1/3,1/3),CV=TRUE)
fit2
pfit2=fit2$posterior
pfit2
ct2 <- table(fit2$class,T)
ct2
```

10.3 Linear Discriminant Analysis: LDA

The following commands reproduce Table 10.2 associated with Example 10.5.1.

```
rm(list = ls())
cush <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1.DAT",
  sep="",col.names=c("Type","Tetra","Preg"))
attach(cush)
cush
#summary(cush)
TL = log(Tetra)
PL = log(Preg)
T=factor(Type)

library(MASS)
fit <- lda(T ~ TL + PL,prior=c(1/3,1/3,1/3))
fit
pfit=predict(fit)
pfit
ct <- table(pfit$class,T)
ct

fit2 <- lda(T ~ TL + PL,prior=c(1/3,1/3,1/3),CV=TRUE)
fit2
pfit2=fit2$posterior
pfit2
ct2 <- table(fit2$class,T)
ct2
```

10.4 Cross-Validation

Commands for cross-validation were contained in the two previous sections.

For *K*-fold cross validation look up the following which relies on the association between discrimination and one-way MANOVA as discussed in the next section.

```
# K-fold cross-validation
library(DAAG)
cv.lm(df=mydata, fit, m=3) # 3 fold cross-validation
```

10.5 Discussion

No computing

10.6 Stepwise LDA

Does the `step` command apply to LDA?

To find the summary statistics by species, use the following.

```
rm(list = ls())
beetles <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\Examp12-2-2.dat",
sep="",col.names=c("y1","y2","y3","y4","y5","y6","s"))
attach(beetles)
beetles
Spec=factor(s)
r12=y1/y2
y=cbind(y1,y2,y3,y4,y5,y6,r12)

colMeans(y[Spec==1,])
cov(y[Spec==1,],y[Spec==1,])
colMeans(y[Spec==2,])
cov(y[Spec==2,],y[Spec==2,])
colMeans(y[Spec==3,])
cov(y[Spec==3,],y[Spec==3,])
```

To compute the statistics for the forward stepwise discriminant analysis, use the following.

```
rm(list = ls())
beetles <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\Examp12-2-2.dat",
sep="",col.names=c("y1","y2","y3","y4","y5","y6","s"))
attach(beetles)
beetles
Spec=factor(s)
r12=y1/y2
y=cbind(y1,y2,y3,y4,y5,y6,r12)

#Step 1:  Look at the F statistics for Spec
beat <- lm(r12 ~ Spec)
anova(beat)
beat <- lm(y1 ~ Spec)
```

```
anova(beat)
beat <- lm(y2 ~ Spec)
anova(beat)
beat <- lm(y3 ~ Spec)
anova(beat)
beat <- lm(y4 ~ Spec)
anova(beat)
beat <- lm(y5 ~ Spec)
anova(beat)
beat <- lm(y6 ~ Spec)
anova(beat)

#Step 2:  Look at the F statistics for Spec
beat <- lm(y1 ~ r12 + Spec)
anova(beat)
beat <- lm(y2 ~ r12 + Spec)
anova(beat)
beat <- lm(y3 ~ r12 + Spec)
anova(beat)
beat <- lm(y4 ~ r12 + Spec)
anova(beat)
beat <- lm(y5 ~ r12 + Spec)
anova(beat)
beat <- lm(y6 ~ r12 + Spec)
anova(beat)

#Step 3:  Look at the F statistics for Spec
beat <- lm(y1 ~ r12 + y4 + Spec)
anova(beat)
beat <- lm(y2 ~ r12 + y4 + Spec)
anova(beat)
beat <- lm(y3 ~ r12 + y4 + Spec)
anova(beat)
beat <- lm(y5 ~ r12 + y4 + Spec)
anova(beat)
beat <- lm(y6 ~ r12 + y4 + Spec)
anova(beat)
```

Further steps follow similarly.

Constructing Table 12.5 (SL 10.4). Desired numbers are in Column “Wilks” and row “Spec”.

```
rm(list = ls())
beetles <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\Examp12-2-2.dat",
sep="",col.names=c("y1","y2","y3","y4","y5","y6","s"))
```

```
attach(beetles)
beetles
Spec=factor(s)
r12=y1/y2
y=cbind(y1,y2,y3,y4,y5,y6,r12)


# Enties from bottom up
# Relevant output is the Wilks statisic for Spec
ya=cbind(y2,y3,y4,y5,y6,r12)
beat <- lm(ya ~ Spec)
anova(beat,test = c("Wilks"))


ya=cbind(y2,y3,y4,y5,r12)
beat <- lm(ya ~ Spec)
anova(beat,test = c("Wilks"))
```

The remaining entries in the Table follow the same pattern except that the entry for r_{12} comes from fitting the univariate one-way ANOVA on Spec with $\Lambda_{obs} = SSE/[SSE + SS(Spec)]$ and with the P value being that from testing Spec.

10.7 Linear Discrimination Coordinates

```
rm(list = ls())
resp <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM10-2.dat",
   sep="",col.names=c("yy1","yy2","yy3","yy4","Drug"))
attach(resp)
resp
D=factor(Drug)
library(MASS)
re <- lda(D ~ yy1+yy2+yy3+yy4)
re
rep = predict(re)
rep

# Eigenvectors are are not unique.
# The following commands transform the MASS
# output to agree with the book.
# It is not important to do this!
YA1 = (-1.9245)*rep$x[,1]
YA2 = (-1.9245)*rep$x[,2]
```

```
# Relabel X axis.  Makes the plots look nicer.
Drr=c("Placebo","Placebo","Placebo","Placebo","Placebo",
"Placebo","Placebo","Placebo","Placebo","Placebo",
"A","A","A","A","A","A","A","A","A","A",
"B","B","B","B","B","B","B","B","B","B")
Dr=factor(Drr)


# Create plots
#dotchart(YA1,groups=Dr)
par(mfrow=c(1,2))
plot(Drug,YA1,xlab="Drug Index")
plot(Dr,YA1,ylab="YA1")
plot(Drug,YA2,xlab="Drug Index")
plot(Dr,YA2,ylab="YA1")
par(mfrow=c(1,1))
#dotchart(YA2,groups=Dr)
plot(YA2[D==1],YA1[D==1],pch=3,ylim=c(-9,8),xlim=c(-5,8),
  xlab="YA2",ylab="YA1")
lines(YA2[D==2],YA1[D==2],pch=16,type="p")
lines(YA2[D==3],YA1[D==3],pch=22,type="p")
legend("bottomright",c("Placebo","A","B"),pch=c(3,16,22))
```

An alternate way to do the linear discrimination.

```
y=cbind(yy1,yy2,yy3,yy4)
#a matrix with individual dependent variables as columns.
y
re <- lda(y,D)
```

10.7.1 Discrimination plot

```
library(ellipse)

rm(list = ls())
b1=0
b2=0
d=1
b3=-.8320503*d
b4=.5547002*d
A = matrix(c(1,1.2,1.2,2),2,2, dimnames = list(NULL, c("b1", "b2")))
E <- ellipse(A, centre = c(b1, b2), t = .95, npoints = 100)
E1 <- ellipse(A, centre = c(b3, b4), t = .95, npoints = 100)
```

```
plot(E,type = 'l',ylim=c(-1.5,2),xlim=c(-2,1.5),xlab=expression(y[1]),
 ylab=expression(y[2]),main="Discriminant Analysis")
lines(E1,type="l",lty=1)
text((b1+.1),(b2-.15),expression(mu[1]),lwd=1,cex=1)
lines(b1,b2,type="p",pch=16)
text((b3+.1),(b4-.15),expression(mu[2]),lwd=1,cex=1)
lines(b3,b4,type="p",pch=19)
```

10.7.2 Discrimination Coordinates

Copied from ANREG commands

Get H and E from a one-way manova. Perhaps from `fitted.values` or `residuals` and `df.residuals` from lm or

```
rm(list = ls())
b1=0
b2=0
d=1
b3=-.8320503*d
b4=.5547002*d
A =.5* matrix(c(1,1.2,1.2,2),2,2, dimnames = list(NULL, c("b1", "b2")))
E <- ellipse(A, centre = c(b1, b2), t = .95, npoints = 100)
E1 <- ellipse(A, centre = c(b3, b4), t = .95, npoints = 100)

The following plots 20 random data points over the ellipse.
library(mvtnorm)
T= rmvnorm(20,c(b1,b2),A)
plot(T,type = 'p',ylim=c(-1.5,2),xlim=c(-2,1.5),xlab=expression(y[1]),
 ylab=expression(y[2]),main="Discriminant Analysis")
T1= rmvnorm(20,c(b3,b4),A)
lines(T1,type="p",pch=4)

TT=rbind(T,T1)
```

10.8 Linear Discrimination

10.9 Modified Binary Regression

10.9.1 Linearly Inseparable Groups: Logistic discrimination, LDA, and SVM

First we produce *ALM-III* Figure 10.6. In the book I treated Carcinoma as the “1” group and bilateral hyperplasia as the “0” group. In the code, that is reversed.

```
rm(list = ls())
cush <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1c.DAT",
  sep="",col.names=c("Type","Tetra","Preg"))
attach(cush)
cush
TL = log(Tetra)
PL = log(Preg)
Tp=Type-2
Tp2=3-Type
# Tp2 is 1 for Bilat hyp and 0 for Carcin

ac2 = glm(Tp2 ~ TL + PL,family=binomial)
summary(ac2)
anova(ac2)
post=c(Type,ac2$fit,1-ac2$fit)
PropProb=matrix(post,15,3,dimnames =
        list(NULL,c("Group","B","C")))
PropProb
# Without adjustment,
# logistic regression uses prior probabilities
# proportional to sample sizes.

# Estimated posterior probabilities
prior=.5 #prior probability for first group (coded as 1)
n=length(Tp)
n1=10  #number of observations in first group
n2=n-n1
PostOdds=(ac2$fit/(1-ac2$fit))*(n2/n1)*(prior/(1-prior))
PostProb=PostOdds/(1+PostOdds)
posttab=c(Type,PostProb,1-PostProb)
PosteriorTable=matrix(posttab,15,3,dimnames =
        list(NULL,c("Group","B","C")))
PosteriorTable
```

```
# Plot of points and discrimination curves
xx2=c(8.3, 3.8, 3.9, 7.8, 9.1, 15.4,  7.7, 6.5, 5.7, 13.6)
yy2=c(1.00, .20, .60, 1.20, .60, 3.60, 1.60, .40, .40, 1.60)
xx3=c(10.2,  9.2,  9.6, 53.8, 15.8)
yy3=c(6.40, 7.90, 3.10, 2.50, 7.60)

x2=log(xx2)
x3=log(xx3)

y2=log(yy2)
y3=log(yy3)
plot(x2,y2, pch=16, ylab="log(Pregnanetriol)",
 ylim=c(-2,3),xlim=c(1,4.5),xlab="log(Tetrahydrocortisone)")
points(x3, y3, pch=22)
legend("bottomright",c("Bilateral Hyperplasia","Carcinoma"),
        pch=c(16,22))

x1=seq(1,4.5,.01)
b=ac2$coef
y2=(b[1]+log(n2/n1)+log(prior/(1-prior))+b[2]*x1)/-b[3]
lines(x1,y2,type="l")
legend("topright",c("Logistic","LDA","SVM"),lty=c(1,2,5))

T=factor(Type)

library(MASS)
fit <- lda(T ~ TL + PL,prior=c(1/2,1/2))
fit
summary(fit)
pfit=predict(fit)
pfit
ct <- table(pfit$class,T)
ct

md=fit$means[1,]+fit$means[2,]
bb=fit$scaling
yLDA=(log(prior/(1-prior))-.5*sum(md*bb)+bb[1]*x1)/-bb[2]
lines(x1,yLDA,type="l",lty=2)



#install.packages("e1071")
library(e1071)
```

```
fit <- svm(T ~ TL + PL,kernel="linear",scale=F)

fit$SV
fit$coefs
fit$rho
x1=seq(1,4.5,.005)
b=t(fit$SV)%*%fit$coefs
#solve <(x1,ySVM)'b>=fit$rho
ySVM=(-fit$rho+log(prior/(1-prior))+b[1]*x1)/-b[2]
# = (fit$rho - b[1]*x1)/b[2]
lines(x1,ySVM,type="l",lty=5)
```

10.9.1.1 *ANREG-II*, Subsection 21.9: Modified

ANREG-II, Subsection 21.9 uses a loglinear model to perform logistic discrimination for all three groups in the Cushing Syndrome data. To double check my earlier logistic two-group regression work, I modified the *ANREG-II* code to handle just 2 groups: Bilateral hyperplasia and Carcinoma.

```
rm(list = ls())
cush <- read.table("C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1c.DAT",
  sep="",col.names=c("Syn","Tetra","Preg"))
attach(cush)
cush

#Create a 2 x 15 table of 0-1 entries,
#each row has 1's for a different type of syndrome
j=rep(seq(1,15),2)
i=c(rep(1,15),rep(2,15))
Tet=c(Tetra,Tetra)
Pre=c(Preg,Preg)
y=c(Syn,Syn)
y[1:10]=1
y[11:15]=0
y[16:25]=0
y[26:30]=1
datal=c(y,i,j,Tet,Pre)
datl=matrix(datal,30,5,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[1:15]/10
p2=ld$fit[15:30]/5
# Produce table
estprob = c(Syn,p1,p2)
EstProb=matrix(estprob,15,3,dimnames =
        list(NULL,c("Group","B","C")))
EstProb

# Table 21.13 Proportional prior probabilities.
post = c(Syn,ld$fit)
PropProb=matrix(post,15,3,dimnames =
        list(NULL,c("Group","B","C")))
PropProb

# Table 21.13 Equal prior probabilities.
p=p1+p2
pp1=p1/p
pp2=p2/p
post = c(Syn,pp1,pp2)
EqProb=matrix(post,15,3,dimnames =
        list(NULL,c("Group", "B","C")))
EqProb
```

10.9.2 Quadratically Separable Groups: Logistic discrimination, QDA, and SVM

We now produce *ALM-III* Figures 10.7 and 10.8. The only difference is in the choice of the tuning parameter for SVM and the SVM fits are unchanged from *ALM-III* Figures 10.3 and This begins a four part program. It creates the data plot and draws the QDA discrimination lines. First I borrowed the plot of the data points from *ANREG-II* and added a legend

10.9.2.1 Enter and plot data

Entering the data:

```
xx2=c(8.3, 3.8, 3.9, 7.8, 9.1, 15.4, 7.7, 6.5, 5.7, 13.6)
yy2=c(1.00, .20, .60, 1.20, .60, 3.60, 1.60, .40, .40, 1.60)
xx3=c(10.2,  9.2,  9.6, 53.8, 15.8)
yy3=c(6.40, 7.90, 3.10, 2.50, 7.60)
x2=log(xx2)
x3=log(xx3)
y2=log(yy2)
y3=log(yy3)
plot(x2,y2, pch=16, ylab="log(Pregnanetriol)",
 ylim=c(-2,3),xlim=c(1,4.5),xlab="log(Tetrahydrocortisone)")
points(x3, y3, pch=22)
legend("bottomright",c("Bilateral Hyperplasia","Carcinoma"),
        pch=c(16,22))
legend("topleft",c("Logistic","QDA","SVM"),lty=c(1,2,5))
```

Reading in the data:

```
rm(list = ls())
cush <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1c.DAT",
  sep="",col.names=c("Type","Tetra","Preg"))
attach(cush)
cush
T=factor(Type)
TL = log(Tetra)
PL = log(Preg)
plot(TL[T==2], PL[T==2], pch=16, ylab="log(Pregnanetriol)",
 ylim=c(-2,3),xlim=c(1,4.5),xlab="log(Tetrahydrocortisone)")
points(TL[T==3], PL[T==3], pch=22)
legend("bottomright",c("Bilateral Hyperplasia","Carcinoma"),
        pch=c(16,22))
legend("topleft",c("Logistic","QDA","SVM"),lty=c(1,2,5))
```

10.9.2.2 Quadratic Logistic Discrimination

```
TL2=TL*TL
PL2=PL*PL
TPL=PL*TL
Tp=Type-2
Tp2=3-Type
# Tp2 is 1 for Bilat hyp and 0 for Carcin
```

```
ac2 = glm(Tp2 ~ TL + PL + TL2 + PL2 + TPL,family=binomial)
summary(ac2)
anova(ac2)
post=c(Type,ac2$fit,1-ac2$fit)
PropProb=matrix(post,15,3,dimnames =
        list(NULL,c("Group","B","C")))
PropProb
#Without adjustment, logistic regression uses prior probabilities
#proportional to sample sizes.

# Estimated posterior probabilities
prior=.5 #prior probability for first group (coded as 1)
n=length(Tp)
n1=10  #number of observations in first group
n2=n-n1
# Correct for sample sizes
PostOdds=(ac2$fit/(1-ac2$fit))*(n2/n1)*(prior/(1-prior))
PostProb=PostOdds/(1+PostOdds)
posttab=c(Type,PostProb,1-PostProb)
PosteriorTable=matrix(posttab,15,3,dimnames =
        list(NULL,c("Group","B","C")))
PosteriorTable

#solve quadratic equation
x1=seq(1,4.5,.1)
bb=ac2$coef
prior=.5
c=log(n2/n1)+log(prior/(1-prior))+bb[1]+bb[2]*x1+bb[4]*x1^2
b= bb[3] + bb[6]*x1
a=bb[5]
delta=b^2-4*a*c
yLR = (-b+sqrt(delta))/(2*a)
yLR2 = (-b-sqrt(delta))/(2*a)
lines(x1,yLR,type="l",lty=1)
lines(x1,yLR2,type="l",lty=1)
```

10.9.2.3 Quadratic Discriminant Analysis

This is a continuation of the previous program.

```
T=factor(Type)
library(MASS)
fit <- qda(T ~ TL + PL,prior=c(.5,.5))
fit
```

```
summary(fit)
pfit=predict(fit)
pfit

#simplify notation.
# Qi is a nonsymmetric square root of
# the sample covariance matrix in group i
mu1=fit$means[1,]
mu2=fit$means[2,]
Q1=fit$scaling[,,1]
Q2=fit$scaling[,,2]
Q1=t(Q1)
Q2=t(Q2)
Q1m1=Q1%*%mu1
Q2m2=Q2%*%mu2

# define and solve quadratic equation.
# add curves to previous plot
prior=.5
c=log(prior/(1-prior)) -(fit$ldet[2] - fit$ldet[1]) +
     .5*(sum(Q2m2*Q2m2)-sum(Q1m1*Q1m1)) +
      (sum(Q1m1*Q1[,1]) - sum(Q2m2*Q2[,1]))*x1 +
     .5*(sum(Q2[,1]*Q2[,1])- sum(Q1[,1]*Q1[,1]))*x1^2
b= (sum(Q1m1*Q1[,2]) - sum(Q2m2*Q2[,2])) +
     (sum(Q2[,1]*Q2[,2])- sum(Q1[,1]*Q1[,2]))*x1
a=.5*(sum(Q2[,2]*Q2[,2]) - sum(Q1[,2]*Q1[,2]))

delta=b^2-4*a*c
yQDA = (-b+sqrt(delta))/(2*a)
yQDA2 = (-b-sqrt(delta))/(2*a)
lines(x1,yQDA,type="l",lty=2)
lines(x1,yQDA2,type="l",lty=2)
```

10.9.2.4 Quadratic SVM

This is a continuation of the program for . Typically you would want to have `scale=T` in `svm`, which is the default. *This is exactly the same SVM program as given earlier.*

```
#install.packages("e1071")
library(e1071)
T=factor(Type)
# define SVM and display outputs
fit <- svm(T ~ TL + PL,kernel="polynomial",degree=2,
        gamma=1,coef0=1,scale=F)
```

```
# Next line reduces tuning parameter, i.e., increases cost.
#fit <- svm(T ~ TL + PL,kernel="polynomial",degree=2,
        gamma=1,coef0=1,scale=F,cost=100)
fit$SV
fit$coefs
fit$rho
fit$fitted # shows the groups that each case was allocated to
# fitted  establishes that my parabolic solution is reasonable.
# predict(fit)

# define and solve quadratic equation.
# add curves to previous plot
# must solve matrix equation that involves inner products
x1=seq(1,4.5,.01)
w=fit$coefs
c=-fit$rho + sum(w)+2*sum(w*fit$SV[,1])*x1+
        sum(w*(fit$SV[,1])^2)*x1^2
b=2*sum(w*fit$SV[,2]) + 2*sum(w*fit$SV[,1]*fit$SV[,2])*x1
a=sum(w*(fit$SV[,2])^2)

delta=b^2-4*a*c
ySVM = (-b+sqrt(delta))/(2*a)
ySVM2 = (-b-sqrt(delta))/(2*a)
lines(x1,ySVM,type="l",lty=5)
lines(x1,ySVM2,type="l",lty=5)


# A plot that svm provides
plot(fit,cush)
```

Chapter 11
Dimension Reduction

https://cran.r-project.org/web/views/Multivariate.html contains an overview of R procedures for multivariate analysis.

11.1 Theory

This code is for showing an ellipse along with its major and minor axes. While the code is correct, you have to futz with the axes lengths in R or latex to get the orthogonal axes to actually look perpendicular. The main thing is that that in R, since the plot appears as a square, the lengths of the plotted *x* and *y* axes have to be the same to get the vectors to look approximately orthogonal. However, latex also allows you to stretch the axes, so you again need the figure's height and width to be the same. But even then, if you want the axes to really look orthogonal, you need to futz a bit.

```
library(ellipse)
rm(list = ls())
b1=1
b2=2
A = matrix(c(1,.9,.9,2),2,2, dimnames=list(NULL, c("b1","b2")))
A
E <- ellipse(A,centre = c(b1, b2),t=.95, npoints=100)

b=seq(0,1.52,.01)
x=1+.507128*b
y=2+.8618708*b
bq=seq(-.65,0,.01)
x1=1+(.8618708*bq)
y1=2-(.507128*bq)

K=5
```

```
bb1=1+(.8618708*K)
bb2=2-(.507128*K)
AA = matrix(c(1,.9,.9,2),2,2, dimnames=list(NULL, c("bb1","bb2")))
AA
EE <- ellipse(AA,centre = c(bb1, bb2),t=.95, npoints=100)

bb=seq(0,1.52,.01)
x=1+.507128*bb
y=2+.8618708*bb
bb=seq(-.65,0,.01)
x1=1+(.8618708*bb)
y1=2-(.507128*bb)



plot(E,type = 'l',ylim=c(.5,3.5),xlim=c(-.5,2.5),
     xlab=expression(y[1]),
 ylab=expression(y[2]),main="Principal Components")
text((b1+.01),(b2-.1),expression(mu),lwd=1,cex=1)
lines(EE,type="l",lty=1)
lines(b1,b2,type="p",pch=19)
lines(x,y,type="l",lty=1)
lines(x1,y1,type="l",lty=1)
text((b1+.5),(b2+.53),expression(a[1]),lwd=1,cex=1)
text((b1-.3),(b2+.12),expression(a[2]),lwd=1,cex=1)

# The following plots 20 random data points over the ellipse.
library(mvtnorm)
T= rmvnorm(20,c(1,2),A)
lines(T[,1],T[,2],type="p")
```

11.1.1 Normal Density Plot for PA-V

```
#install.packages("ellipse")
#Do this only once on your computer
library(ellipse)

b1=1
b2=2
A = matrix(c(1,.9,.9,2),2,2, dimnames = list(NULL, c("b1", "b2")))
A
E <-  ellipse(A,centre = c(b1, b2),t=.95,npoints = 100)
E1 <- ellipse(A,centre = c(b1, b2),t=.5 ,npoints = 100)
```

```
E2 <- ellipse(A,centre = c(b1, b2),t=.75,npoints = 100)

plot(E,type = 'l',ylim=c(.5,3.5),xlim=c(0,2),
     xlab=expression(y[1]),
 ylab=expression(y[2]),main="Normal Density")
text((b1+.01),(b2-.1),expression(mu),lwd=1,cex=1)
lines(E1,type="l",lty=1)
lines(E2,type="l",lty=1)
lines(b1,b2,type="p",lty=3)
```

11.2 Sample Principal Components

There is a section of commands in the R code for *ANREG-II* for doing principal component regression.

There are three data files available for the turtle shell data: combined, females, and males. This example uses the males.

```
rm(list = ls())
turtm <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM10-3m.dat",
sep="",col.names=c("L","W","H"))
attach(turtm)
turtm
x3=10^(3/2)*log(L)
x2=10^(3/2)*log(W)
x1=10^(3/2)*log(H)
n=length(L)
X = matrix(c(x1,x2,x3),n,3)
S=var(X)
S
R=cor(X)
R
# two equivalent ways of specifying the data
fit <- prcomp(~ x1+x2+x3, scale=F)
summary(fit)
fit$rotation
fit <- prcomp(X, scale=F)
summary(fit)
fit$rotation


e <- eigen(S,symmetric=TRUE)
e$values
```

```
e$vectors
PC <- X %*% e$vec  #Matrix of principal component scores
```

11.2.1 Using Principal Components

```
rm(list = ls())
b1=1
b2=2
A = matrix(c(1,.9,.9,2),2,2, dimnames=list(NULL, c("b1","b2")))
A
E <- ellipse(A,centre = c(b1, b2),t=.95, npoints=100)

b=seq(0,1.52,.01)
x=1+.507128*b
y=2+.8618708*b
bq=seq(-.65,0,.01)
x1=1+(.8618708*bq)
y1=2-(.507128*bq)

K=-.8
bb1=1+(.507128*K)
bb2=2-(.8618708*K)
AA = matrix(c(1,.9,.9,2),2,2, dimnames=list(NULL, c("bb1","bb2")))
AA
EE <- ellipse(AA,centre = c(bb1, bb2),t=.95, npoints=100)

bb=seq(0,1.52,.01)
x=1+.507128*bb
y=2+.8618708*bb
bb=seq(-.65,0,.01)
x1=1+(.8618708*bb)
y1=2-(.507128*bb)



plot(E,type = 'l',ylim=c(0,5),xlim=c(-1,2.5),
     xlab=expression(y[1]),
 ylab=expression(y[2]),main="Principal Components")
lines(b1,b2,type="p",pch=19)
text((b1+.02),(b2-.15),expression(mu[1]),lwd=1,cex=1)
lines(EE,type="l",lty=1)
lines(bb1,bb2,type="p",pch=19)
text((bb1-.1),(bb2),expression(mu[2]),lwd=1,cex=1)
```

11.2.2 Kernel PCA

11.3 Multidimensional Scaling

You have some measure of the distances between items and you want to find where these points are located relative to one another. We are going to reconsider the school test data on which we did factor analysis. We don't have the data on the 225 students but we do have the correlation matrix which is something like the opposite of a distance matrix. Correlations are large when things are close and small when they are far away.

11.3.1 Classical MDS

First we do the examples in the book. Then do an extra example. Then do other forms of multidimensional scaling.

Example 11.3.1

```
rm(list = ls())
cush <-read.table(
        "C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1.DAT",
  sep="",col.names=c("Syn","Tetra","Preg"))
attach(cush)
cush
ID = list("a1","a2","a3","a4","a5","a6",
"b1","b2","b3","b4","b5","b6","b7","b8","b9","b10",
"c1","c2","c3","c4","c5")
TL=log(Tetra)
PL=log(Preg)

CS=cbind(TL,PL)
CSd=dist(CS)  # default is

CSfit <- cmdscale(CSd,eig=TRUE, k=2)
CSfit


x <- CSfit$points[,1]
```

```
y <- CSfit$points[,2]
plot(y, x, xlab="Coordinate 2", ylab="Coordinate 1",
  main="Classical Multidimensional Scaling", type="n")
text(y, x, labels = ID, cex=.9)
```

Example 11.3.2

```
rm(list = ls())

ID = list("Gaelic","English","History",
        "Arithmetic","Algebra","Geometry")
R = matrix(c(1.000,0.439,0.410,0.288,0.329,0.248,
0.439,1.000,0.351,0.354,0.320,0.329,
0.410,0.351,1.000,0.164,0.190,0.181,
0.288,0.354,0.164,1.000,0.595,0.470,
0.329,0.320,0.190,0.595,1.000,0.464,
0.248,0.329,0.181,0.470,0.464,1.000),6,6)
#R=-log(R)
R2=R*R
D=1-R
D2=1-R2
par(mfrow=c(2,1))
Dfit <- cmdscale(D,eig=TRUE, k=2)
Dfit
x <- Dfit$points[,1]
y <- Dfit$points[,2]
plot(x, y, xlab="Coordinate 1", ylab="Coordinate 2",
xlim=c(-.46,.35),ylim=c(-.26,.26),
main="CMDS:  Correlations", type="n")
text(x, y, labels = ID, cex=.9)

D2fit <- cmdscale(D2,eig=TRUE, k=2)
D2fit
x2 <- D2fit$points[,1]
y2 <- D2fit$points[,2]
plot(x2, y2, xlab="Coordinate 1", ylab="Coordinate 2",
xlim=c(-.4,.46),ylim=c(-.46,.46),
main="CMDS:  Squared Correlations", type="n")
text(x2, y2, labels = ID, cex=.9)
```

11.3.1.1 Application to Heart Rate Data

Find distances from MANOVA heart rate data and plot CMDS coordinates.

```
rm(list = ls())
resp <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM10-2.dat",
   sep="",col.names=c("Y1","Y2","Y3","Y4","Drug"))
attach(resp)
resp
Y=cbind(Y1,Y2,Y3,Y4)

ID=seq(1,30)
Yd=dist(Y)  # default is

Yfit <- cmdscale(Yd,eig=TRUE, k=2) # k is the number of dim
Yfit
x <- Yfit$points[,1]
y <- Yfit$points[,2]
plot(x, y, xlab="Coordinate 1", ylab="Coordinate 2",
  main="Metric MDS", type="n")
text(x, y, labels = ID, cex=.9)
```

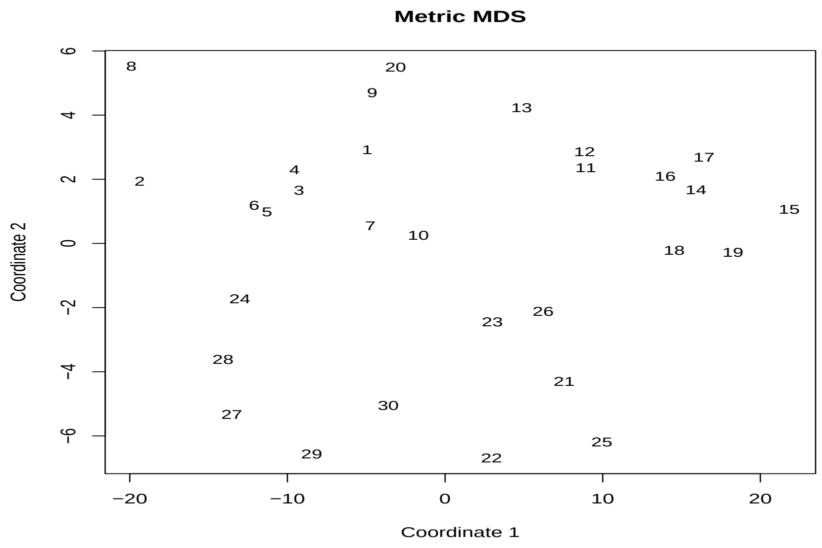


Fig. 11.1 Multidimensional Scaling: Heart Data.

11.3.2 Nonmetric MDS

This is not discussed in the text!

For cases in which "distance" is nothing more than an ordering. Generally, a building 10 miles away is twice as far away as a building that is 5 miles away. A true distance measure has that property, but often we want to apply multidimensional scaling to "discrepencies" rather than distances. The program below uses the Classical solution as a starting point.

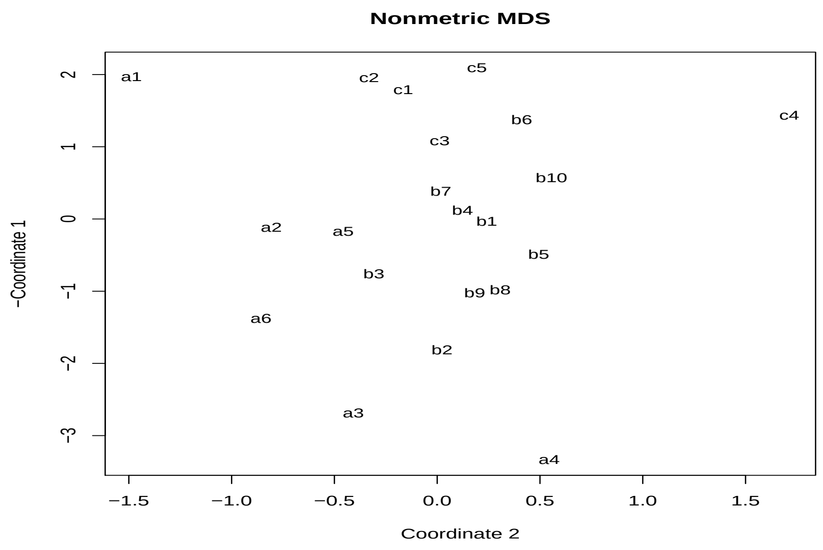


Fig. 11.2 Nonmetric Multidimensional Scaling: Cushing Syndrome Data.

```
rm(list = ls())
cush <-
read.table("C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1.DAT",
  sep="",col.names=c("Syn","Tetra","Preg"))
attach(cush)
cush
ID = list("a1","a2","a3","a4","a5","a6",
"b1","b2","b3","b4","b5","b6","b7","b8","b9","b10",
"c1","c2","c3","c4","c5")
TL=log(Tetra)
PL=log(Preg)

CS=cbind(TL,PL)
CSd=dist(CS)  # default is
```

```
library(MASS)
CSfit <- isoMDS(CSd,k=2) # k is the number of dim
CSfit


x <- CSfit$points[,1]
y <- CSfit$points[,2]
plot(y, -x, xlab="Coordinate 2", ylab="-Coordinate 1",
  main="Nonmetric MDS", type="n")
text(y, -x, labels = ID, cex=.9)

rm(list = ls())
resp <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM10-2.dat",
   sep="",col.names=c("Y1","Y2","Y3","Y4","Drug"))
attach(resp)
resp
Y=cbind(Y1,Y2,Y3,Y4)

ID=seq(1,30)
Yd=dist(Y)  # default is
library(MASS)
Yfit <- isoMDS(Yd,k=2) # k is the number of dim
Yfit
x <- Yfit$points[,1]
y <- Yfit$points[,2]
plot(x, y, xlab="Coordinate 1", ylab="Coordinate 2",
  main="Nonmetric MDS", type="n")
text(x, y, labels = ID, cex=.9)
```

11.3.3 Example 11.3.2

This is our factor analysis example on correlations between 6 tests. All we have is a correlation matrix.

$$R = \begin{bmatrix} 1.000 & 0.439 & 0.410 & 0.288 & 0.329 & 0.248 \\ 0.439 & 1.000 & 0.351 & 0.354 & 0.320 & 0.329 \\ 0.410 & 0.351 & 1.000 & 0.164 & 0.190 & 0.181 \\ 0.288 & 0.354 & 0.164 & 1.000 & 0.595 & 0.470 \\ 0.329 & 0.320 & 0.190 & 0.595 & 1.000 & 0.464 \\ 0.248 & 0.329 & 0.181 & 0.470 & 0.464 & 1.000 \end{bmatrix}.$$

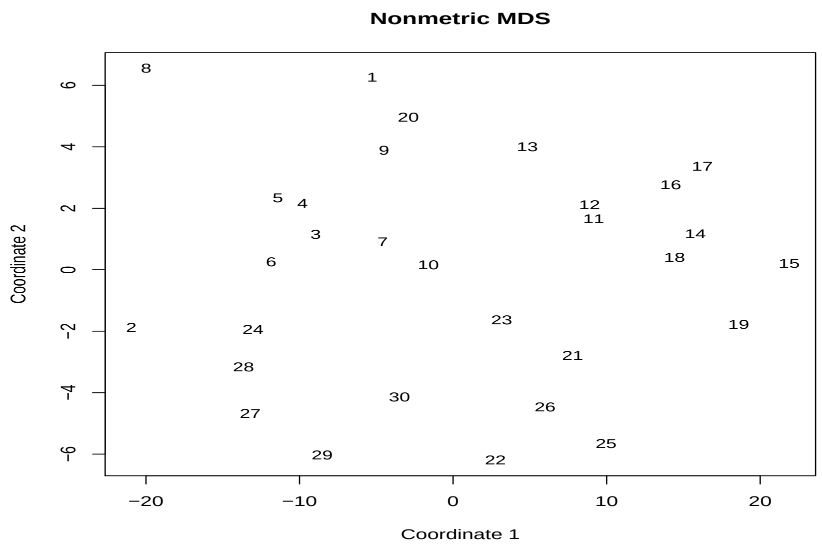


Fig. 11.3 Nonmetric Multidimensional Scaling: Heart Data.

Not looking at how far apart the 225 people are, looking at how far apart the 6 tests are. Y would be 6×225 with 225 objects measured on every item of interest.

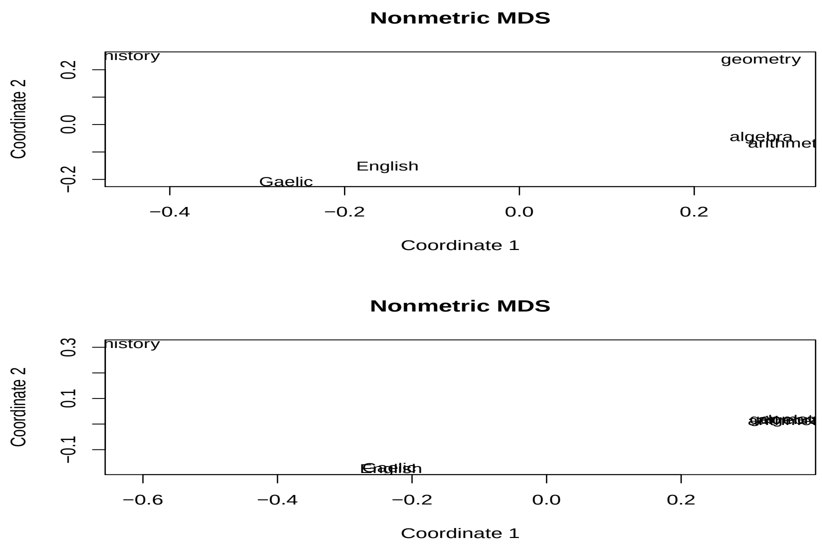


Fig. 11.4 Multidimensional Scaling: Test Data, $D = 1 - R$.

11.3.3.1 Code

CMDS

```
rm(list = ls())

ID = list("Gaelic","English","History",
        "Arithmetic","Algebra","Geometry")
R = matrix(c(1.000,0.439,0.410,0.288,0.329,0.248,
0.439,1.000,0.351,0.354,0.320,0.329,
0.410,0.351,1.000,0.164,0.190,0.181,
0.288,0.354,0.164,1.000,0.595,0.470,
0.329,0.320,0.190,0.595,1.000,0.464,
0.248,0.329,0.181,0.470,0.464,1.000),6,6)
#R=-log(R)
R2=R*R
D=1-R
D2=1-R2
par(mfrow=c(2,1))
Dfit <- cmdscale(D,eig=TRUE, k=2)
Dfit
x <- Dfit$points[,1]
y <- Dfit$points[,2]
plot(x, y, xlab="Coordinate 1", ylab="Coordinate 2",
xlim=c(-.46,.35),ylim=c(-.26,.26),
main="CMDS:  Correlations", type="n")
text(x, y, labels = ID, cex=.9)

D2fit <- cmdscale(D2,eig=TRUE, k=2)
D2fit
x2 <- D2fit$points[,1]
y2 <- D2fit$points[,2]
plot(x2, y2, xlab="Coordinate 1", ylab="Coordinate 2",
xlim=c(-.4,.46),ylim=c(-.46,.46),
main="CMDS:  Squared Correlations", type="n")
text(x2, y2, labels = ID, cex=.9)
```

Other approaches.

```
library(MASS)
Rfit <- isoMDS(R,k=2) # k is the number of dim
Rfit


x <- Rfit$points[,1]
y <- Rfit$points[,2]
plot(x, y, xlab="Coordinate 1", ylab="Coordinate 2",
  main="Nonmetric MDS", type="n")
text(x, y, labels = ID, cex=.9)
```

11.4 Data Compression

Singular value decomposition as data compression.

Reproduce a black and white photo gray scale matrix from singular value decomposition.

It looks like the `twitter.png` file below no longer exists so I have substituted one of my own. My .png files have much larger dimensions. His seem to be $28 \times 28 \times 4$. Also, I have had no luck with `download`ing my `.png` files from the internet (as indicated in the program) and getting them recognized. If I read them from my hard drive, I have no problem.

No idea if any of this works. I borrowed it from Dr. Achim Zeileis but I no longer remember how or when. Achim Zeileis, Department of Statistics, Faculty of Economics and Statistics, Universität Innsbruck, Austria. https://eeecon.uibk.ac.at/~zeileis/

```
download.file(
#"http://www.greenmountaindiapers.com/skin/common_files/
#modules/Socialize/images/twitter.png",destfile = "twitter.png")
"http://www.stat.unm.edu/~fletcher/m10.png", destfile="m10.png")
library("png")
library("colorspace")
#x <- readPNG("twitter.png")
x <- readPNG("m10.png")
dim(x)
## [1] 28 28  4

#Now we can transform this into various other formats:
#y is a vector of hex character strings,
#specifying colors in R. yg is the corresponding
#desaturated color (again as hex character)
#with grayscale only. yn is the numeric amount #of gray.
#All three objects are arranged into 28 x 28 matrices at the end
y <- rgb(x[,,1], x[,,2], x[,,3], alpha = x[,,4])
yg <- desaturate(y)
yn <- col2rgb(yg)[1, ]/255
dim(y) <- dim(yg) <- dim(yn) <- dim(x)[1:2]
I hope that at least one of these versions is what you are looking for.
#To check the pixel matrices I have written a
#small convenience function for visualization:
```

```
pixmatplot <- function (x, ...) {
  d <- dim(x)
  xcoord <- t(expand.grid(1:d[1], 1:d[2]))
  xcoord <- t(xcoord/d)
  par(mar = rep(1, 4))
  plot(0, 0, type = "n", xlab = "", ylab = "", axes = FALSE,
    xlim = c(0, 1), ylim = c(0, 1), ...)
  rect(xcoord[, 2L] - 1/d[2L], 1 - (xcoord[, 1L] - 1/d[1L]),
    xcoord[, 2L], 1 - xcoord[, 1L], col = x, border = "transparent")
}
#For illustration let's look at:
pixmatplot(y)
pixmatplot(yg)
```

The following is an R package for image processing: https://dahtah.github.io/imager/imager.html.

11.5 Factor Analysis

An overview of R procedures for multivariate analysis is available at https://cran.r-project.org/web/views/Multivariate.html.

11.5.1 Terminology and Applications

This section uses only the base function `factanal`. A subsection of the next section uses the facilities from the library `psych`.

The example presents maximum likelihood estimation from correlation matrix (rather than raw data). Unrotated factor loadings.

```
R = matrix(c(1.000,0.439,0.410,0.288,0.329,0.248,
0.439,1.000,0.351,0.354,0.320,0.329,
0.410,0.351,1.000,0.164,0.190,0.181,
0.288,0.354,0.164,1.000,0.595,0.470,
0.329,0.320,0.190,0.595,1.000,0.464,
0.248,0.329,0.181,0.470,0.464,1.000),6,6)
test=factanal(covmat=R,n.obs=220,factors=2,rotation="none")
test
test$unique
test$loadings[,1:2]
```

Illustration of maximum likelihood estimation from raw data: male turtle shell data with one factor. **Not in book.**

```
rm(list = ls())
turtm <-
read.table("C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM10-3m.dat",
sep="",col.names=c("L","W","H"))
attach(turtm)
turtm
x3=10^(3/2)*log(L)
x2=10^(3/2)*log(W)
x1=10^(3/2)*log(H)
n=length(L)
X = matrix(c(x1,x2,x3),n,3)
test=factanal(X,factors=1,rotation="none")
test=factanal(~x1+x2+x3,factors=1,rotation="none")
```

11.5.2 Maximum Likelihood Theory

The following illustrates the varimax rotation and the three plots for the “test” data. `factanal` does not do quartimax, but the next subsubsection uses a different program that allows quartimax rotation.

```
plot(test$loadings[,1],test$loadings[,2],
        xlim=c(-1,1),ylim=c(-1,1),
        type="n",xlab="Factor 1",ylab="Factor 2")
x=seq(-1,1,.01)
lines(x,0*x)
lines(0*x,x)
text(test$loadings[,1],test$loadings[,2],
labels=c(1,2,3,4,5,6))


test=factanal(covmat=R,n.obs=220,factors=2,
        rotation="varimax")
#  varimax is the default rotation
test
test$unique
test$loadings[,1:2]
plot(test$loadings[,1],test$loadings[,2],
     xlim=c(-1,1),ylim=c(-1,1),
     type="n",xlab="Factor 1",ylab="Factor 2")
lines(x,0*x)
lines(0*x,x)
text(test$loadings[,1],test$loadings[,2],
     labels=c(1,2,3,4,5,6))
```

```
# I read the quartimax loadings from the book
# factanal does not do quartimax
f1=c(0.260,0.344,0.111,0.777,0.731,0.580)
f2=c(0.650,0.536,0.587,0.139,0.184,0.188)
test$loadings=matrix(c(f1,f2),6,2)
test$loadings
plot(test$loadings[,1],test$loadings[,2],
     xlim=c(-1,1),ylim=c(-1,1),
     type="n",xlab="Factor 1",ylab="Factor 2")
lines(x,0*x)
lines(0*x,x)
text(test$loadings[,1],test$loadings[,2],
     labels=c(1,2,3,4,5,6))
```

The above code to plot two factors will not work on the turtle data because with only $q = 3$ variables, `factanal` will only fit one factor.

11.5.2.1 Psych-ed out

Everything done thus far and more can be done using the library `psych`. To get the quartimax rotation, another library is also needed.

```
R = matrix(c(1.000,0.439,0.410,0.288,0.329,0.248,
0.439,1.000,0.351,0.354,0.320,0.329,
0.410,0.351,1.000,0.164,0.190,0.181,
0.288,0.354,0.164,1.000,0.595,0.470,
0.329,0.320,0.190,0.595,1.000,0.464,
0.248,0.329,0.181,0.470,0.464,1.000),6,6)


#install.packages("psych")
library(psych)
#install.packages("GPArotation")
library(GPArotation)

fit <- fa(R, fm="ml",nfactors=2,n.obs=220,
     rotate="none")
fit
fit <- fa(R, fm="ml",nfactors=2,n.obs=220,
     rotate="varimax")
fit
fit <- fa(R, fm="ml",nfactors=2,n.obs=220,
      rotate="quartimax")
fit
```

```
fit <- fa(R, fm="pa",nfactors=2, rotate="none")
```

11.5.3 ??

Still need to check these out.

```
# Determine Number of Factors to Extract
library(nFactors)
ev <- eigen(cor(mydata)) # get eigenvalues
ap <- parallel(subject=nrow(mydata),var=ncol(mydata),
  rep=100,cent=.05)
nS <- nScree(x=ev$values, aparallel=ap$eigen$qevpea)
plotnScree(nS)


# PCA Variable Factor Map
library(FactoMineR)
result <- PCA(mydata) # graphs generated automatically
```

Chapter 12
Clustering

12.1 Pointwise Distance Measures

For an $n \times q$ data matrix Y, compute Euclidean distances using `dist(Y)`. $\mathbf{L}^1$ distances are obtained from `dist(Y,"manhattan")`. $\mathbf{L}^p$ distances are obtained using for, say, $p = 1.5$ the command `dist(Y,"minkowski",p=1.5)`. $\mathbf{L}^\infty$ distances are obtained using `"maximum"`. A couple of other options are also available. Apparently the library `ecodist` computes Mahalanobis squared distances with `distance(Y,"mahal")`. The `dist` command creates an object of the `dist` type that is required for input into the hierarchical clustering program `hclust` used in Section 2.

In the code below I use the Cushing's syndrome data to illustrate getting the Euclidean distance matrix (`CSd`), the $\mathbf{L}^1$ distances, and also construct the Mahalanobis squared distance matrix (`D`) using the command `mahalanobis` and then turn `D` into a `dist` object.

```
rm(list = ls())
cush <-
read.table("C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1.DAT",
  sep="",col.names=c("Syn","Tetra","Preg"))
attach(cush)
cush
ID = list("a1","a2","a3","a4","a5","a6",
"b1","b2","b3","b4","b5","b6","b7","b8","b9","b10",
"c1","c2","c3","c4","c5")
TL=log(Tetra)
PL=log(Preg)

# Euclidean distance matrix
CS=cbind(TL,PL)
CSd=dist(CS)
CSd
```

```
dist(CS,"manhattan")

# setup for computing Mahal
n=length(TL)
S=cov(CS)
# create dummy D matrix
d=seq(1,n*n)
D=matrix(d,n)

# compute Mahal sq. distance matrix
for(i in 1:n)
{
D[i,] = mahalanobis(CS,CS[i,],S)
}
# Turn distance matrix into R object of class "dist"
D=as.dist(D)
```

Alternatively, to get Mahalanobis, I think that you can, (1) find $S^{-1} = QQ'$, (2) transform the data matrix Y into YQ, and (3) compute Euclidean distances on the transformed matrix. To find a matrix Q you can use

```
dc <- svd(S)
phi <- dc$d
D <- diag(sqrt(1/phi))
P <- dc$u
Q=P %*% D %*% t(P)
```

12.2 Hierarchical Cluster Analysis

Having found the distance matrix as in the previous section, the actual clustering is done in `hclust`. Specifying the linkage method is straightforward except that R seems to think that the old `ward.D` is wrong and the new version `ward.D2` is correct. According to Murtagh and Legendre (2014), it is merely that `ward.D` requires inputs that are *Euclidean* squared distances while `ward.D2` takes inputs that are *Euclidean* distances.

```
par(mfrow=c(2,1))
CS1 =hclust(CSd, method = "single")
plot(CS1,labels=ID,main=NULL)
groups <- cutree(CS1, k=5) # cut tree into 5 clusters
rect.hclust(CS1, k=5, border="red")

CS1 =hclust(CSd, method = "complete")
plot(CS1,labels=ID,main=NULL)
```

```
groups <- cutree(CS1, k=5) # cut tree into 5 clusters
rect.hclust(CS1, k=5, border="red")

par(mfrow=c(2,1))
CS1 =hclust(CSd, method = "average")
plot(CS1,labels=ID,main=NULL)
groups <- cutree(CS1, k=5) # cut tree into 5 clusters
rect.hclust(CS1, k=5, border="red")

CS1 =hclust(CSd, method = "centroid")
plot(CS1,labels=ID,main=NULL)
groups <- cutree(CS1, k=5) # cut tree into 5 clusters
rect.hclust(CS1, k=5, border="red")

# These 2 Ward methods should be the same
par(mfrow=c(2,1))
CS1 =hclust(CSd, method = "ward.D2")
plot(CS1,labels=ID)
groups <- cutree(CS1, k=5) # cut tree into 5 clusters
rect.hclust(CS1, k=5, border="red")


CS1 =hclust(CSd^2, method = "ward.D")
plot(CS1,labels=ID)
groups <- cutree(CS1, k=5) # cut tree into 5 clusters
rect.hclust(CS1, k=5, border="red")


par(mfrow=c(1,1))
# Inputing distances to ward.d gives Ward
# based on root distances
CS1 =hclust(CSd, method = "ward.D")
plot(CS1,labels=ID)
groups <- cutree(CS1, k=5) # cut tree into 5 clusters
rect.hclust(CS1, k=5, border="red")
```

12.3 K-means Clustering

The kmeans program uses the raw data as input rather than the distance matrix and seems to only use Euclidean distances. It might make sense to scale the data before apply the algorithm or even transform it so that Euclidean distances on the transformed data are Mahalanobis distances on the original data, see the end of Section 1.

```
rm(list = ls())
cush <-
 read.table("C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM12-1.DAT",
  sep="",col.names=c("Syn","Tetra","Preg"))
attach(cush)
cush
ID = list("a1","a2","a3","a4","a5","a6",
"b1","b2","b3","b4","b5","b6","b7","b8","b9","b10",
"c1","c2","c3","c4","c5")
TL=log(Tetra)
PL=log(Preg)


CS=cbind(TL,PL)
fit <- kmeans(CS, 5)
# get cluster means
aggregate(CS,by=list(fit$cluster),FUN=mean)
# append cluster assignment
Clust=fit$cluster
data.frame(cbind(TL,PL,ID,Clust))
data.frame(cbind(ID[order(Clust)],Clust[order(Clust)]))
```

The default algorithm is `algorithm = "Hartigan-Wong"`. Other choices are `"MacQueen"`, `"Lloyd"`, `"Forgy"` although the last two names define the same algorithm.

12.4 Spectral Clustering

Cluster the principal components.

Appendix A
Matrices

Commands for matrix algebra where given at the beginning of Chapter 2.

A.8 Eigenvalues; Eigenvectors

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.

The command `svd` also works for nonsquare matrices.

Appendix B
Three-Factor ANOVA

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.

```
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)


#Summary tables
A=factor(a)
B=factor(b)
C=factor(c)
b2=b*b
sabc <- lm(y ~ A:B:C)
coef=summary(sabc)
coef
anova(sabc)


sabc <- lm(y ~ A*B*C)
#coef=summary(sabc)
#coef
anova(sabc)


sabc <- lm(y ~ A:B + C-1)
```

```
#coef=summary(sabc)
#coef
anova(sabc)


sabc <- lm(y~A/(b+I(b^2))+C -1)
#coef=summary(sabc)
#coef
anova(sabc)

sabc <- lm(y ~ A + A:b + C - 1)
coef=summary(sabc)
coef
anova(sabc)


sabc <- lm(y ~ A/b + C - 1)
coef=summary(sabc)
coef
anova(sabc)

# Generate the variable A2 discussed in the text.
A2=A
A2[(A2 == 3)]  <- 2


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

B.0.1 Computing

Because it is the easiest program I know, most of the analyses in *ANREG-I* 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 B.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. 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.

Table B.1 Moisture data, indices, and predictors.

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

An R script for fitting $[AB][C]$ follows. R needs to locate the data file, which in this case is located at `E:\Books\ANREG2\DATA2\tab16-1.dat`.

```
scheffe <- read.table("E:\\Books\\ANREG2\\DATA2\\tab16-1.dat",
sep="",col.names=c("y","a","b","c"))
attach(scheffe)
scheffe
summary(scheffe)

#Summary tables
A=factor(a)
B=factor(b)
C=factor(c)
X=b
X2=X*X
sabc <- lm(y ~ A:B + C)
coef=summary(sabc)
coef
anova(sabc)
```

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;
```

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

Appendix C
MANOVA

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


# 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.

C.1 Notation

Multivariate data

$$Y_{n\times q} = [Y_1, \dots, Y_q] = \begin{bmatrix} y_1' \\ \vdots \\ y_n' \end{bmatrix}.$$

The y_is are independent but $\text{Cov}(y_i) = \Sigma$.

Multivariate linear model as a collection of univariate models,

$$Y_h = X\beta_h + e_h, \quad \text{E}(e_h) = 0, \quad \text{Cov}(e_h) = \sigma_{hh}I, \quad h = 1, \dots, q. \tag{C.1.1}$$

Combine these into a multivariate model

$$[Y_1, \dots, Y_q] = X[\beta_1, \dots, \beta_q] = [e_1, \dots, e_q]$$

or

$$Y = XB + e \tag{C.1.2}$$

Similarly to Y write,

$$X_{n\times p} = [X_1, \dots, X_p] = \begin{bmatrix} x_1' \\ \vdots \\ x_n' \end{bmatrix},$$

$$e_{n\times q} = [e_1, \dots, e_q] = \begin{bmatrix} \varepsilon_1' \\ \vdots \\ \varepsilon_n' \end{bmatrix}.$$

Write the multivariate model in terms of individuals,

$$y_i' = x_i'B + \varepsilon_i', \quad \text{E}(\varepsilon_i) = 0, \quad \text{Cov}(\varepsilon_i) = \Sigma, \quad i = 1, \dots, n. \tag{C.1.3}$$

Change X and B notation. J is an n vector of 1s,

$$X_{n\times p} = [J, X_1, \dots, X_{p-1}] = [J, X_*].$$

$$B = \begin{bmatrix} \mu' \\ B_* \end{bmatrix}$$

$$XB = [J, X_*]\begin{bmatrix} \mu' \\ B_* \end{bmatrix} = J\mu' + X_*B_*$$

In terms of individuals $i = 1, \ldots, n$, the model $y_i' = x_i'B + \varepsilon_i'$ becomes,

$$y_i' = \mu' + x_{*i}'B_* + \varepsilon_i' \tag{C.1.4}$$

Appendix D
Neural Networks and Deep Learning as Nonparametric/Nonlinear Regression

The R neural net programs that we will consider are neuralnet and nnet. We also use the nonlinear least squares program nls.

After writing this I became aware of another R NN program neuralnetwork in a package `ANN2` (anomaly detection NN). It easily incorporates ridge and LASSO penalties. There is also a package/program nls2 that will fit nonlinear regression with a grid search. The package nlsr has a nonlinear regression program `nlxb` that may deal better with collinearity. Another nonlinear regression option is the package gslnls

The machine learning "task view" `https://cran.r-project.org/web/views/MachineLearning.html` mentions other programs (and does not mention `neuralnet`).

I wonder how steepest descent algorithms with random starting points would work with overparameterized linear models.

D.1 Univariate Neural Networks

We present code for running `neuralnet` on the battery data.

EXAMPLE D.1.1. *Battery Data.*
This particular example is for $D = 3$, $r = 2$ as indicated by the `neuralnet` option `hidden=c(2,2)`, which is a $D-1$ vector containing the value of r for each level. The default fitting algorithm is resilient backpropagation with weight backtracking.

```
## Clear R and read the data
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
```

```
battery

##Run neuralnet
library(neuralnet)
batneu = neuralnet(y ~ x,hidden=c(2,2),act.fct =
    "logistic",data=battery,linear.output=T)

## Plot NN model diagram with estimates
plot(batneu)
## Additional neuralnet output commands
# batneu
# summary(batneu)
#batneu$generalized.weight
#batneu$result.matrix

## Plot data and fitted values
## compute R^2
yhat=batneu$net.result[[1]]
R2=cor(yhat,y)^2
plot(x,y,main=bquote(R^2== .(round(R2, 5))))
lines(x,yhat)


## extract and print the Bk matrices and beta vector
## from the batneu$weights list of lists
B1=batneu$weights[[1]][[1]]
B2=batneu$weights[[1]][[2]]
beta=batneu$weights[[1]][[3]]
B1
B2
beta
batneuwts=c(B1,B2,beta)
```

The last part of the code is only for $D = 3$. If you define D in the code, the estimated β is always,

```
beta=batneu$weights[[1]][[D]]
```

The number of B_k matrices to be extracted depends on D. I was not ambitious enough to write code that would work for any D,

The book defines

$$\tilde{\beta} \equiv [\beta', \text{Vec}(B_{D-1})', \ldots, \text{Vec}(B_1)]'$$

but NN software tends to list things in reverse order,

$$B_1, B_2, \ldots, B_{D-1}, \beta.$$

In particular my code from above, `batneuwts=c(B1,B2,beta)`, equals $[\text{Vec}(B_1)', \text{Vec}(B_2)', \beta']$. For $D = 2$ my code would be `batneuwts=c(B1,beta)` $= [\text{Vec}(B_1)', \beta]$ and can be used as starting values for `nnet`, which expects its starting values in this order.

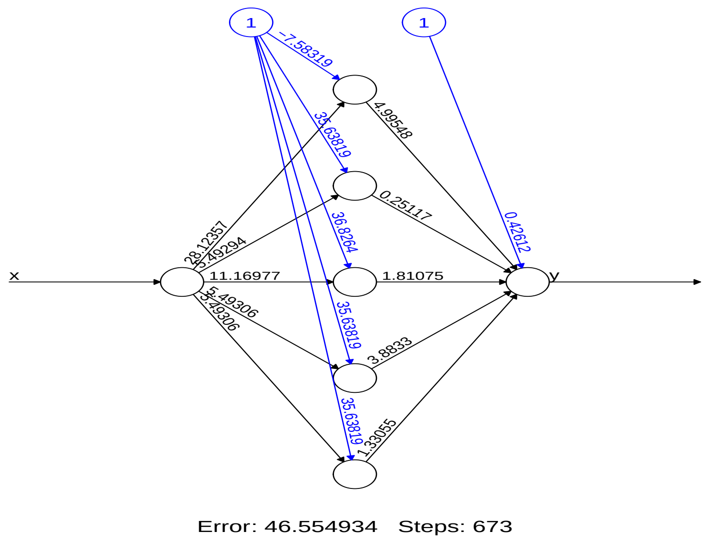


Fig. D.1 Battery data neural net fits: $D = 2$, $r = 5$.

The `plot(batneu)` command gives the NN diagram and parameter estimates. Figure D.1 here provides another example with `hidden=5`, or equivalently `hidden=c(5)`. □

D.1.1 Relation to Spanning Functions

No computing.

D.2 Nonlinear Regression

Ideally, `nls` should be able to fit any NN. While `nls` *might* have been written with the idea that the model would be identifiable and have unique least squares estimates, it should take a starting value and search for a local minimim to $SSE(\tilde{\beta})$. This should be true even for the nonidentifiable models assoociated with $r \geq 2$. As discussed in the book, collinearity seems to be a bigger problem. When $r = 1$, there seems to be no reason to doubt the identifiability of the NN models but it does not seem particularly easy to prove that they are identifiable.

EXAMPLE D.2.1. This is the only case ($D = 2$, $r = 1$) where we can easily compare all three of `neuralnet`, `nnet`, and `nls`. The code introduces running both `nnet` and `nls`.

The book presented parameter estimates from four neuralnet and nnet runs as given below. They are presented here with their names from the code that generated them. The estimates `batnnet$wts` are `nnet` estimates ($\tilde{\beta}$ rearranged as discussed earlier). The `neuralnet` estimates ($\tilde{\beta}$ rearranged) are accumulated into `batneuwts`. The estimates seem to have converged but to slightly different values.

```
> batnnet$wts
[1] -7.685779 28.468946  7.711415  4.984809
> batneuwts
[1] -7.594703 28.163069  7.702761  4.994409
> batnnet$wts
[1] -7.684596 28.463871  7.711285  4.985015
> batneuwts
[1] -7.598368 28.173712  7.703679  4.993587
> batnnet$wts
[1] -7.687263 28.472109  7.711415  4.984930
> batneuwts
[1] -7.599077 28.177550  7.703560  4.993619
> batnnet$wts
[1] -7.684437 28.463444  7.711252  4.985046
> batneuwts
[1] -7.582689 28.122157  7.701728  4.995634
```

Rerunning the code should give similar, but not exactly the same values (due to random starting values).

To get similar values, run the following code four times.

```
## Clear R and read the data
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

##Run neuralnet
library(neuralnet)
batneu = neuralnet(y ~ x,hidden=c(1),act.fct =
    "logistic",data=battery,linear.output=T)
```

```
## extract and print the Bk matrices and beta vector
## from the batneu$weights list of lists
B1=batneu$weights[[1]][[1]]
beta=batneu$weights[[1]][[2]]
B1
beta
batneuwts=c(B1,beta)

## Run nnet
library(nnet)
batnnet = nnet(y ~ x,size=1,linout=T)

#summary(batnnet)
## Print out both sets of weights for visual comparison
batnnet$wts
batneuwts
```

The next code shows that for $D = 2$, $r = 1$, `nls` converges to the `nnet` solution, essentially immediately. The sets of starting values are copied from the book. Similar results are obtained when not specifying starting values, i.e. by eliminating the `start=c()` option from `nls`.

```
## Clear R and read the data
#rm(list = ls())
#par(mfrow=c(4,2))
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)


betatilde=c(-7.582689, 28.122157,  7.701728,  4.995634)
betatilde=c(-7.684437, 28.463444,  7.711252,  4.985046)
batnls = nls(y ~ beta1
+ beta2 * (exp(b111 + b112*x)/  (1+exp(b111+b112*x))),
start=c(beta1=betatilde[3],beta2=betatilde[4],
b111=betatilde[1],b112=betatilde[2])
)
#summary(batnls)
batnls

#coef(batnls)
#coef(summary(batnls))
#fitted(batnls)
```

```
#anova(batnls)
```

This `nls` code also works and has the convenience of using the logistic function.

```
\begin{verbatim}
## Clear R and read the data
#rm(list = ls())
#par(mfrow=c(4,2))
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)


betatilde=c(-7.582689, 28.122157,  7.701728,  4.995634)
betatilde=c(-7.684437, 28.463444,  7.711252,  4.985046)
logistic <- function(w)
{
lg = exp(w)/(1+exp(w))
return(lg)}

batnls = nls(y ~ beta1
+ beta2 * logistic(b111 + b112*x),
start=c(beta1=betatilde[3],beta2=betatilde[4],
b111=betatilde[1],b112=betatilde[2])
)
#summary(batnls)
# Parameters listed in order of appearance
batnls
yhatnls=predict(batnls,x)
R2=cor(yhatnls,y)^2
plot(x,y,main=c(R2))
lines(x,yhatnls)
```

□

EXAMPLE D.2.2. $D = 3, r = 1$.
I programmed $D = 3, r = 1$ which is an identifiable nonlinear regression. The following code establishes that even with good starting values obtained from `neuralnet`, `nls` cannot find an answer due, usually, to a "singular gradient" but occasionally due to "Error in numericDeriv ... Missing value or an infinity produced when evaluating the model" or the step factor getting below the minimum. Run it multiple times to see the various error messages.

```
## Clear R and read the data
rm(list = ls())
```

```
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

logistic <- function(w)
{
lg = exp(w)/(1+exp(w))
return(lg)}

##Run neuralnet
library(neuralnet)
h=c(1,1)
D=length(h)+1
batneu = neuralnet(y ~ x,hidden=h,act.fct =
    "logistic",data=battery,linear.output=T)

## extract B1, B2 and beta vector from the
## batneu$weights list of lists
# for(k in 1:D){
B1=batneu$weights[[1]][[1]]
B1
B2=batneu$weights[[1]][[2]]
B2
beta=batneu$weights[[1]][[D]]
beta

batnls = nls(y ~ beta1
+ beta2 * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=beta[1],beta2=beta[2],b211=B2[1,1],
b221=B2[2,1],b111=B1[1,1],b121=B1[2,1])
)
```

`nls` uses a default Gauss-Newton algorithm but allows other options. In my experience, the other options do not work any better for this problem. The package and program `nls2` has additional algorithms including a grid search.

The reason `nls` is not converging is probably because the linear model approximation to the nonlinear model is displaying severe collinearity. The nonlinear model has the approximating linear model

$$Y - F(\tilde{\beta}_0) = W(\tilde{\beta}_0)\delta + e, \qquad W(\tilde{\beta}_0) \equiv \mathbf{d}_{\tilde{\beta}} F(\tilde{\beta}_0) \equiv [J, W_*].$$

The severe collinearity is demonstrated by the following code that gives the eigenvalues of the covariance matrix computed from the "data matrix" W_*. Here $\tilde{\beta}_0$ is being determined by a `neuralnet` run. I determined the nature of the $W(\cdot)$ function and programmed it. In fact, columns 3 and 4 of W are highly collinear which makes even back propagation difficult.

```
## Clear R and read the data
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

# define logistic function
logistic <- function(w)
{
lg = exp(w)/(1+exp(w))
return(lg)}


# define derivative of the logistic function
dlogistic <- function(w)
{
dlg = exp(w)/((1+exp(w))^2)
return(dlg)}


##Run neuralnet to get starting values
library(neuralnet)
h=c(1,1)
D=length(h)+1
batneu = neuralnet(y ~ x,hidden=h,act.fct =
    "logistic",data=battery,linear.output=T)

## extract B1, B2 and beta vector from the
## batneu$weights is a list of lists
B1=batneu$weights[[1]][[1]]
B1
B2=batneu$weights[[1]][[2]]
B2
beta=batneu$weights[[1]][[D]]
beta

# construct
```

```
betatilde=c(beta,B2,B1)
betatilde
# to reprodice book, use
# betatilde=c(5.2801397,7.8749274,-0.8069346,
3.5837052,-7.1122252,24.2575694)



# notation for nls starting values
beta1=betatilde[1]
beta2=betatilde[2]
b211=betatilde[3]
b221=betatilde[4]
b111=betatilde[5]   #B1[1,1]
b121=betatilde[6]   #B1[2,1]

# model matrix W for approximating linear model
# W evaluated at neuralnet parameter estimates
W=c(rep(1,41), logistic( b211 + b221*logistic(b111 + b121*x)),
beta2*dlogistic( b211 + b221*logistic(b111 + b121*x)),
beta2*dlogistic( b211 + b221*logistic(b111 + b121*x))*
   logistic(b111 + b121*x),
beta2*dlogistic( b211 + b221*logistic(b111 + b121*x))*
   b221*dlogistic(b111 + b121*x),
beta2*dlogistic( b211 + b221*logistic(b111 + b121*x))*
   b221*dlogistic(b111 + b121*x)*x)
W=matrix(W,41,6)
# print out the approximate model matrix
W

#Eigenvalues and vectors for the covariance matrix of the
# nonintercept predictors in the
eigen(cov(W[,2:6]))

# This code is for the continuation of the
# example in the Back Propagation subsection.
eigen(cov(W[,3:4]))
eigen(cov(W[,5:6]))
```

As is, this code will not reproduce the book because it generates a "random" $\tilde{\beta}_0$. The book's $\tilde{\beta}_0$ is in the code as a comment. □

D.2.0.1 Other nls fits.

I wrote down nls code for $D = 3$, $r = 2$ just to get my head around how to write nls models. It employs the logistic function that I coded elsewhere.

```
batnls = nls(y ˜ beta1
+ beta2 * logistic( b211 + b212*logistic(b111 + b112*x)
    +  b213 * logistic(b121  + b122*x))
+ beta3 * logistic(b221 + b222*logistic(b111 + b112*x)
    +  b223 * logistic(b121  + b122*x))
)
```

I then used this as a basis for simpler models that I fitted (or not, because they failed to run). In particular, I used this as the basis for programming `nls` to fit $D = 2$, $r = 2$ which is a nonidentifiable nonlinear regression. I used starting values obtained from `neuralnet`. Once again, `nls` couldn not find an answer due, usually, to a "singular gradiant".

```
## Clear R and read the data
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

##Run neuralnet
library(neuralnet)
h=c(2)
D=length(h)+1
batneu = neuralnet(y ˜ x,hidden=h,act.fct =
    "logistic",data=battery,linear.output=T)

## extract B1, B2 and beta vector from the
## batneu$weights list of lists
# for(k in 1:D){
B1=batneu$weights[[1]][[1]]
B1

beta=batneu$weights[[1]][[D]]
beta
plot(batneu)

logistic <- function(w)
{
lg = exp(w)/(1+exp(w))
return(lg)}


batnls = nls(y ~ beta1
+ beta2 * logistic(b111 + b121*x) + beta3 * logistic(b112 + b122*x),
start=c(beta1=beta[1],beta2=beta[2],beta3=beta[3],b111=B1[1,1],
b121=B1[2,1],b112=B1[1,2],b122=B1[2,2])
)
```

D.2.1 Back Propagation

EXAMPLE D.2.2 CONTINUED. Code for this was given in Example D.2.2.

D.2.1.1 Pseudo back propagation

This is some exploratory work I did with trying to get `nls` Gauss-Newton to work by partitioning the fitting procedure. None of it succeeded in get `nls` to work.

This first code is for illustration and not intended to be run. It illustrates a variation on back propagation. There are three sets of parameters, B_1, B_2, and β. Back propagation holds two of them fixed and finds the third so as to minimize the SSE, so it requires 3 runs to modify all the parameters. In this code we hold only one of them fixed and find the other two. In the practical code that appears later, we hold either B_1 or B_2 fixed, so β gets modified in each run. We did this because `nls` would not run when we fixed β. The code simply shows the calls for the overall nls run and then three calls each with one of the parameters fixed.

```
#neuralnet(x,y,hidden=c(1,1))
batnls = nls(y ~ beta1
+ beta2 * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=beta[1],beta2=beta[2],b211=B2[1,1],
b221=B2[2,1],b111=B1[1,1],b121=B1[2,1])
)


## These next three pieces explore back propagation
## They run nls after fixing some of the parameters
## to equal their neuralnet values.


beta1=beta[1]
beta2=beta[2]
```

```
batnls = nls(y ~ beta1
+ beta2 * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=beta[1],beta2=beta[2],b211=B2[1,1],
b221=B2[2,1],b111=B1[1,1],b121=B1[2,1])
)
batnls


b211=B2[1,1]
b221=B2[2,1]
batnls = nls(y ~ beta1
+ beta2 * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=beta[1],beta2=beta[2],
b111=B1[1,1],b121=B1[2,1])
)
batnls

b111=B1[1,1]
b121=B1[2,1]
batnls = nls(y ~ beta1
+ beta2 * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=beta[1],beta2=beta[2],b211=B2[1,1],b221=B2[2,1])
)
batnls
```

To see that even this code is not converging, run the code and then rerun the code inside the for loop, which runs from a `betatilde` to another `betatilde` and contains a third `betatilde`. The fact that the three `betatilde`s are not converging is the problem. This routine uses the notation $\tilde{\beta}' \equiv (B_1', B_2', \beta')'$

```
## Clear R and read the data
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

logistic <- function(w)
{
lg = exp(w)/(1+exp(w))
return(lg)}

##Run neuralnet
```

```
library(neuralnet)
h=c(1,1)
D=length(h)+1
batneu = neuralnet(y ~ x,hidden=h,act.fct =
    "logistic",data=battery,linear.output=T)

## extract B1, B2 and beta vector from the
## batneu$weights list of lists
# for(k in 1:D){
B1=batneu$weights[[1]][[1]]
B1
B2=batneu$weights[[1]][[2]]
B2
beta=batneu$weights[[1]][[D]]
beta

betatilde=c(B1,B2,beta)



# perform iter steps of pseudo back propagation
iter=40
for(i in 1:iter){

betatilde
b211=betatilde[3] #B2[1,1]
b221=betatilde[4] #B2[2,1]
batnls = nls(y ~ beta1
+ beta2 * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=betatilde[5],beta2=betatilde[6],
b111=betatilde[1],b121=betatilde[2])
)
betatilde[5:6]=coef(batnls)[1:2]
betatilde[1:2]=coef(batnls)[3:4]
betatilde

b111=betatilde[1]   #B1[1,1]
b121=betatilde[2]   #B1[2,1]
batnls = nls(y ~ beta1
+ beta2 * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=betatilde[5],beta2=betatilde[6],
b211=betatilde[3],b221=betatilde[4])
)

betatilde[5:6]=coef(batnls)[1:2]
betatilde[3:4]=coef(batnls)[3:4]
betatilde
}
```

I revised the code to actually do one step of back propagation, i.e, where two of the matrices B_1, B_2, and β are held fixed.

```
## Clear R and read the data
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

##Run neuralnet
library(neuralnet)
h=c(1,1)
D=length(h)+1
batneu = neuralnet(y ~ x,hidden=h,act.fct =
    "logistic",data=battery,linear.output=T)

## extract B1, B2 and beta vector from the
## batneu$weights list of lists
# for(k in 1:D){
B1=batneu$weights[[1]][[1]]
B1
B2=batneu$weights[[1]][[2]]
B2
beta=batneu$weights[[1]][[D]]
beta

logistic <- function(w)
{
lg = exp(w)/(1+exp(w))
return(lg)}


## These next three pieces explore back propagation
## They run nls after fixing some of the parameters
## to equal their neuralnet values.


b211=B2[1,1]
b221=B2[2,1]
```

```
batnls = nls(y ~ beta[1]
+ beta[2] * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=beta[1],beta2=beta[2],b211=B2[1,1],b221=B2[2,1],b111=B1[1,1],
b121=B1[2,1])
)



b211=B2[1,1]
b221=B2[2,1]
batnls = nls(y ~ beta[1]
+ beta[2] * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=beta[1],beta2=beta[2],b111=B1[1,1],
b121=B1[2,1])
)

b111=B1[1,1]
b121=B1[2,1]
batnls = nls(y ~ beta1
+ beta2 * logistic( b211 + b221*logistic(b111 + b121*x)),
start=c(beta1=beta[1],beta2=beta[2],b211=B2[1,1],b221=B2[2,1])
)
```

D.3 Computational Issues

EXAMPLE D.3.1. Program for running $(D=3, r=1)$ 1000 times and computing mean vector and covariance matrix.

```
## Clear R and read the data
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

logistic <- function(w)
{
lg = exp(w)/(1+exp(w))
return(lg)}

YY=rep(1,1000*6)
```

```
YY=matrix(YY,1000,6)

for(i in 1:1000){
##Run neuralnet
library(neuralnet)
h=c(1,1)
D=length(h)+1
batneu = neuralnet(y ~ x,hidden=h,act.fct =
    "logistic",data=battery,linear.output=T)
## extract B1, B2 and beta vector from the
## batneu$weights list of lists
# for(k in 1:D){
B1=batneu$weights[[1]][[1]]
B1
B2=batneu$weights[[1]][[2]]
B2
beta=batneu$weights[[1]][[D]]
beta
YY[i,]=c(beta,B2,B1)
}

cov(YY)
colMeans(YY)
```

It is very obvious from the following covariance matrix that the answers are not consistent.

```
> cov(YY)
           [,1]       [,2]        [,3]       [,4]        [,5]        [,6]
[1,]  0.5753809 -0.4894115  -0.7770350   1.764040  -0.2246971  -0.3047345
[2,] -0.4894115  2.3579861  -0.3857658  -3.043234  -1.3709014   4.4923470
[3,] -0.7770350 -0.3857658   4.9928132 -10.716670   5.7829443  -6.3938296
[4,]  1.7640404 -3.0432339 -10.7166696  36.456578 -12.4099353   7.1189539
[5,] -0.2246971 -1.3709014   5.7829443 -12.409935   8.4313529 -10.5456531
[6,] -0.3047345  4.4923470  -6.3938296   7.118954 -10.5456531  20.6282824
> colMeans(YY)
[1]  5.7503597  6.9913594 -0.5063129 17.9064310 -4.7870357 11.7851015
```

D.3.0.1 Fitting `neuralnet` and feeding into `nnet`

EXAMPLE D.3.2 The code below is for producing figures like Figure D.3. It works for $D = 2$ and any r. Figure D.3 is a 4 by 2 matrix of plots. This code produces two plots, so it needs to be run 4 times. *On the first run*, you should uncomment the `rm` and `par` commands near the top, but for the next 3 runs, they should both be commented. Subsequent sets of 4 runs can follow to you hearts content.

You can pick any value of *r* to put into `neuralnet`'s `hidden` option but then `nnet`'s `size` command needs to take the same *r* value. Since `neuralnet` and `nnet` are in different libraries, you need to run `library()` quite a bit. Within the code ## indicates comments about this code and other valuable options for exploring NNs that are not needed to produce the plot.

```
## Clear R and read the data
#rm(list = ls())
#par(mfrow=c(4,2))
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

##Run neuralnet
library(neuralnet)
batneu = neuralnet(y ~ x,hidden=5,act.fct =
    "logistic",data=battery,linear.output=T)
## Command for specifying starting weights
## within neuralnet()
#startweights = c(1, 1, 1, 1)

## These commands would, if uncommented,
## give the fitted value plot from neuralnet

yhat=batneu$net.result[[1]]
## compute R^2 and NN diagram, if uncommented
R2=cor(yhat,y)^2
#plot(x,y,main=paste0("R^2=", round(R2, 5)))
plot(x,y,main=bquote(R^2== .(round(R2, 5))))
lines(x,yhat)


#plot(batneu)
## Additional neuralnet output
# summary(batneu)
# batneu
#batneu$generalized.weight
#batneu$result.matrix
#predict(batneu,battery)

## extract the B1 matrix and beta vector from the
## batneu$weights list of lists
B1=batneu$weights[[1]][[1]]
```

```
beta=batneu$weights[[1]][[2]]
## create a vector of starting value
## weights for input to nnet
# Note:  batneuwts is betatilde rearranged
batneuwts=c(B1,beta)


## Run nnet using the
## neuralnet output weights "batneuwts"
## as starting values for nnet
library(nnet)
batnnet = nnet(y ~ x,size=5,linout=T,
Wts=batneuwts)

#summary(batnnet)
## Print out both sets of weights for visual comparison
#batnnet$wts
#batneuwts

##Compute square root of sum of squared
##prediction/fitted-value differences between
##neuralnet and nnet and similar for parameter differences
b=c(sqrt(sum((yhat-batnnet$fitted.values)^2)),
sqrt(sum((batneuwts-batnnet$wts)^2)))
## Plot for Fig. D.1
plot(x,y,main=paste0("fit diff=", round(b[1], 3),
   "    " ,"par diff=", round(b[2], 3)))
lines(x,batnnet$fitted.values)
## compute R^2
cor(batnnet$fitted.values,y)^2
```

Use $\tilde{\beta}_1$ and $\tilde{\beta}_2$ to denote vectors of (estimated) parameters from `neuralnet` and `nnet`, respectively. Similarly define fitted value vectors $\hat{Y}_1$ and $\hat{Y}_2$. The numbers over the `nnet` plots are $\|\hat{Y}_1 - \hat{Y}_2\|$ and $\|\tilde{\beta}_1 - \tilde{\beta}_2\|$.

To get a better look at the effects of starting values, you could comment out `plot(x,y,main=c(R2))` and `lines(x,yhat)` *after* the `neuralnet` run, so you get 8 graphs with 2 numbers above them all from `nnet`. That gives 8, instead of 4, looks at how often the procedure goes bad.

Figure D.2 gives an illustration for $D = 2$, $r = 1$ The R^2s are very close indicating the neuralnet fits are very close. $\|\hat{Y}_1 - \hat{Y}_2\|$ and $\|\tilde{\beta}_1 - \tilde{\beta}_2\|$ are small but from looking at examples there are naturally (very) slight differences in the “converged” values of $\tilde{\beta}_1$ and $\tilde{\beta}_2$ but the values they are converging to are consistently (slightly) different.

.

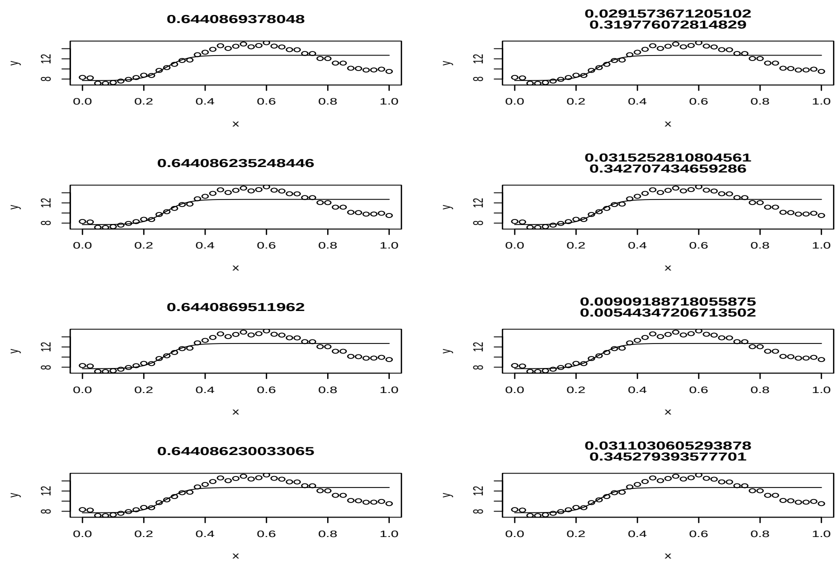


Fig. D.2 Battery data neural net fits: $D = 2$, $r = 1$, neuralnet feeding nnet.

D.3.0.2 Fitting `nnet` and feeding into `neuralnet`

This code produces plots similar to book Figure D.3 except the initial fit is from nnet and those values are used as starting values for neuralnet. Sometimes neuralnet does not change the nnet values and sometimes it changes the nnet converged valued considerably. And that happens regardless of whether the model is being fitted well or not. This was discussed but no figures were presented in the book.

```
## Clear R and read the data
#rm(list = ls())
#par(mfrow=c(4,2))
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery


## Run nnet using the
## neuralnet output weights "batneuwts"
## as starting values for nnet
library(nnet)
batnnet = nnet(y ~ x,size=5,linout=T)
```

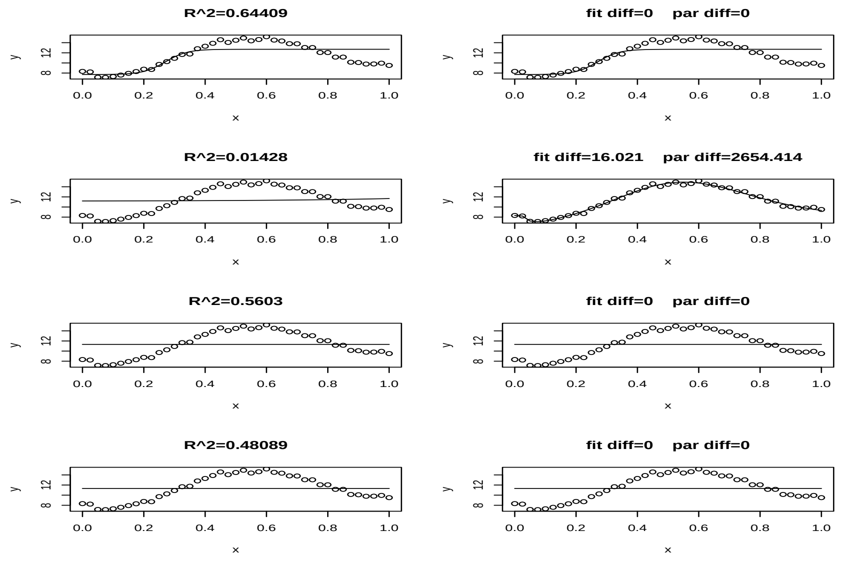


Fig. D.3 Battery data neural net fits: $D = 2$, $r = 5$, nnet feeding neuralnet.

```
## compute R^2 and NN diagram, if uncommented
R2=cor(batnnet$fitted.values,y)^2
plot(x,y,main=paste0("R^2=", round(R2, 5)))
lines(x,batnnet$fitted.values)

##Run neuralnet
library(neuralnet)
batneu = neuralnet(y ~ x,hidden=5,act.fct =
    "logistic",data=battery,linear.output=T,
     startweights = batnnet$wts)
yhat=batneu$net.result[[1]]
## extract the B1 matrix and beta vector from the
## batneu$weights list of lists
B1=batneu$weights[[1]][[1]]
beta=batneu$weights[[1]][[2]]
## create a vector of starting value
## weights for input to nnet
batneuwts=c(B1,beta)

#summary(batnnet)
## Print out both sets of weights for visual comparison
batnnet$wts
```

```
batneuwts

##Compute square root of sum of squared
##prediction/fitted-value differences between
##neuralnet and nnet and similar for parameter differences
b=c(sqrt(sum((yhat-batnnet$fitted.values)^2)),
sqrt(sum((batneuwts-batnnet$wts)^2)))
## Plot for Fig. D.1
plot(x,y,main=paste0("fit diff=", round(b[1], 3),
   "    " ,"par diff=", round(b[2], 3)))
lines(x,yhat)
## compute R^2
#cor(yhat,y)^2
```

D.3.0.3 Miscellany for `neuralnet` with $D > 2$

This is not very different. I was just looking at some `neuralnet` fits with $D = 4$

```
## Clear R and read the data
#rm(list = ls())
#par(mfrow=c(4,2))
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

##Run neuralnet
library(neuralnet)
h=c(1,1,1)
D=length(h)+1
batneu = neuralnet(y ~ x,hidden=h,act.fct =
    "logistic",data=battery,linear.output=T)


yhat=batneu$net.result[[1]]
## compute R^2 and NN diagram, if uncommented
R2=cor(yhat,y)^2
plot(x,y,main=c(R2))
lines(x,yhat)


#plot(batneu)
## Additional neuralnet output
```

```
# summary(batneu)
# batneu
#batneu$generalized.weight
#batneu$result.matrix
#predict(batneu,battery)

## extract the B1 matrix and beta vector from the
## batneu$weights list of lists
# for(k in 1:D){
# B3 will not exist unless h is a 3-vector.
B1=batneu$weights[[1]][[1]]
B1
B2=batneu$weights[[1]][[2]]
B2
B3=batneu$weights[[1]][[3]]
B3
beta=batneu$weights[[1]][[D]]
beta
```

I did some comparisons of how similar the fitted values were when $D = 2,3$.

With $D = 4$, $r = 1$, sometimes the fitted parameter values give an essentially constant fit of $\hat{y}_i = 11.32251$. In particular, these estimates do that.

```
> B1=batneu$weights[[1]][[1]]
> B1
          [,1]
[1,] -1.475321
[2,]  2.045264
> B2=batneu$weights[[1]][[2]]
> B2
         [,1]
[1,] 4.192253
[2,] 2.778969
> B3=batneu$weights[[1]][[3]]
> B3
         [,1]
[1,] 4.157848
[2,] 6.312652
> beta=batneu$weights[[1]][[D]]
> beta
         [,1]
[1,] 6.068829
[2,] 5.253832
```

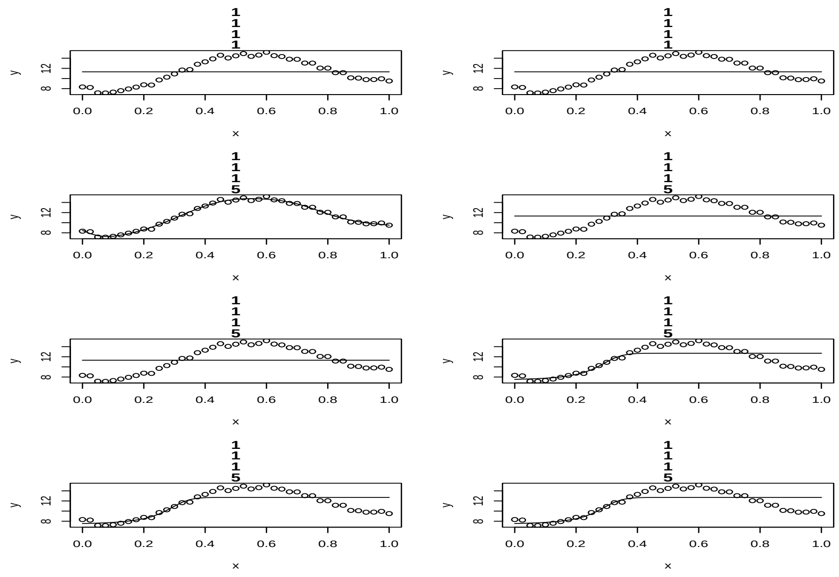


Fig. D.4 Battery data neural net fits: $D = 4$, $r_1 = r_2 = r_3 = 1$, $r_4 = 1, 5$.

D.4 Classification

No illustrations were given in the book but fitting logistic models can be accomplished for 0-1 y data by using the `neuralnet` subcommand/option `linear.output=FALSE` or by the `nnet` subcommand/option `linout=FALSE` (which is the default).

D.5 Generalized Weights

These are a standard part of `neuralnet` output. For the `neuralnet` object `batneu`, they are available as `batneu$generalized.weight`.

```
## Clear R and read the data
rm(list = ls())
battery <- read.table(
"C:\\E-drive\\Books\\LINMOD23\\DATA\\ALM1-1.dat",
header=TRUE,
sep="")#,col.names=c("Case","y","t","x"))
attach(battery)
battery

##Run neuralnet
```

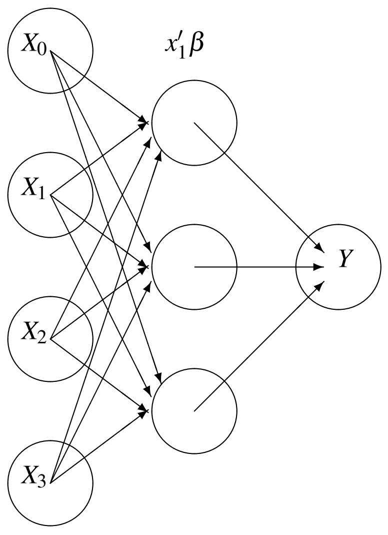
```
library(neuralnet)
batneu = neuralnet(y ~ x,hidden=c(2,2),act.fct =
    "logistic",data=battery,linear.output=T)

batneu$generalized.weight
```

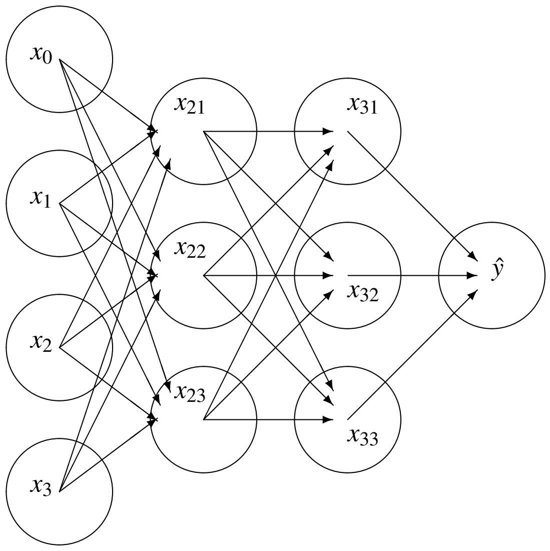
D.6 Multivariate Neural Networks

No computational illustrations.

D.7 Diagrams

My relatively lame attempts to create my own NN diagrams using the `picture` command in LATEX. (Before I found that `neuralnet` happily and easily creates diagrams.)


X_0
$x_1'\beta$
X_1
Y
X_2
X_3


x_0
x_1
x_2
x_3
x_{21}
x_{22}
x_{23}
x_{31}
x_{32}
x_{33}
$\hat{y}$

Index