Notes that are supplementary to

Data Analysis and Graphics Using R – An Example-Based Approach

John Maindonald and John Braun

These will be updated from time to time. They were last modified on September 9, 2006.

Chapter 1

Subsection 1.1.3, page 3

In `plot(ACT ~ Year, data=ACTpop)`, `ACT ~ Year` is a *graphics formula.* Read "Plot `ACT` (on the y-axis) against `Year` (on the x-axis)."

Subsection 1.1.5, page 4

In using `help.search()`, it can help to specify a particular package, e.g.

```
help.search("sort", package="base")
```

Subsection 1.3.2, p.7: Retaining objects between sessions

Workspace management

Use of a single working directory and a single **.RData** image file is likely to be adequate for working through our scripts and exercises, and will for most users be the preferred option as they work through the first two or three chapters. When however several different projects are in progress at the same time, it becomes essential to find some good way to keep separate the different files and R objects. Additionally, crucial files and objects from projects that are for the time being complete will typically be kept for future reference or further investigation.

It therefore makes sense, as soon as readers have made some modest progress in their use of R, to develop and practice a strategy that will be effective for later day to day use. This is especially important for those who, alongside their use of the book, are working on their own R-based projects. We suggest the following strategy:

- As suggested in Subsection 1.3.2, use a separate working directory and associated workspace file for each separate project. For example use a directory called **DAAGR** for work related to this book, and a directory called **project1** for work on the first of several planned R-based projects.
- Impose a second level of structure by the use of multiple workspace image files, created as described in Section 12.9.1. For example, within the directory **DAAGR**, create image files **ch1.RData**, **ch2.RData**, etc., that hold user-created objects that may be required for future use once work on a chapter is for the time being complete.

Following a strategy that uses multiple workspace image files within a single working directory, a user who has finished working through chapter 1 might take the following actions:

```
## First remove objects that are not wanted, or that can easily be
## retrieved, e.g., from an R package.
rm(p, Animals, trees)         # These are from 1.8.3, 1.8.4 & 1.8.5
## Next, save remaining objects in an image file.
save.image(file="ch1.RData")  # Save remaining objects
```

```
## Finally, clear the workspace, and quit.
rm(list=ls())                # Clear the workspace
q()                          # Quit
```

The user will then be asked whether the workspace is to be saved. As the workspace is at this point empty, an affirmative answer will generate a default **.RData** image file in the working directory that is empty of R objects. Thus, when a new session is started in that directory, the workspace will be empty, ready perhaps for the objects that will be loaded or created in the course of working through Chapter 2.

At some later point, to access the objects in **ch1.RData**, alternatives are:

- Use `load("ch1.RData")` to load them all into the workspace.
- Use `attach("ch1.RData")` to place them on the search path.

The attaching of an image file is a useful way to gain access to one or more objects, from a much larger number of objects in a workspace image file in another directory. Clearly the path to that other directory must then be given along with the file name.

Copying objects into the workspace

Technically, the search path is a vector of database names, in which the workspace is placed first. An assignment `y <- x` takes the first object `x` that appears in a database on the search path, and assigns it to an object `y` in the workspace. More generally, `x` can be replaced by an expression. The same name can appear on both sides of the assignment. To copy, e.g., the object `olddata` that is in from an attached image file ("database") into the workspace, type

```
olddata <- olddata
```

If there is an existing object `olddata` in the workspace, it must first be removed, perhaps after making a copy of it.

Saving the command history

In systems that are set up suitably, so that the up arrow can be used to retrieve earlier commands, the command line history is saved from one session to another. The commands `savehistory()` and `loadhistory()` can be used to control this process. By default, the history is saved in the file `.RHistory`.

For explicit saving of the history at the end of every session, put a call to `savehistory()` into the function `.Last`. Thus, if `.Last` does not already exist, type in:

```
.Last <- function()savehistory()
```

Section 1.5

The notation used in this section to extract columns from a data frame can be extended to allow the extraction of any selection of columns from a data frame. For example, the data frame `ACTpop` that we created and used in Section 1.1.3 has a subset of the columns of the data frame `austpop` in DAAG. It can be created thus:

```
ACTpop <- austpop[, c(1,9)]
```

or

```
ACTpop <- austpop[, c("year","ACT")]]
```

The first column of the data frame that is so created has the name `year`, rather than `Year` as in Section 1.1.3.

Subsection 1.8.4 – lattice graphics

Further points to note are:

> The setting of `pointsize`, if supplied when a graphics device is started, is ignored. See `help(trellis.par.set)` for information on how to set the `fontsize` and other lattice parameters. An example is:

```
trellis.par.set(fontsize=list(text=14, points=10))
  # NB: The setting is for the current device only.
```

> For a white background, specify:

```
trellis.par.set(col.whitebg())
  # Again, the setting is for the current device only.
```

Relational and logical operators

These warrant more attention than they receive in Chapters 1 and 12. The notes that follow cover some of the more important points.

Relational Operators (`<`, `<=`, `>`, `>=`, `==` and `!=`)

For details of relational operators, see `help(Comparison)`, `help(Logic)`, and `help(Syntax)`.

Logical operators (`&`, `|`, etc.)

The logical operators `&` (**and**) and `|` (**or**) operate on vectors. They have counterparts `&&` and `||` that expect a single element; if either operand is a vector, the first element only is taken. Thus compare:

```
> c(T, T, F) & c(F, T, F)
[1] FALSE  TRUE FALSE
> c(T, T, F) && c(F, T, F)
[1] FALSE
```

`if` *and* `ifelse()`

Chapter 1 does not discuss `if` statements. The only place where they appear, in the code that we give in the book, is in the discussion of user-written functions in Subsection 12.2.2.

The function `ifelse()` may be regarded as a vector form of the the `if` statement. The comparison is made, and the resultant action taken, for all elements of a vector. See `help(ifelse)`.

Subsection 1.8.2: Identification and labeling of points on the figure region

More generally, the `identify()` function may be given a set of labels that will be plotted upon clicking the left mouse button. Try for example:

```
attach(primates}
plot(Bodywt, Brainwt)
identify(Bodywt, Brainwt, row.names(primates))
## Terminate by clicking outside the plot region,
## or perhaps by clicking inside the region, but
## with a button other than the first.
detach(primates)
```

Section 1.9 – Functions

User-written functions appear in a relatively small number of places in this book. Though important, they are not central to our exposition. Readers whose primary interest is in data analysis should be able to learn what they need as occasion demands.

Section 1.7 had an example of an inline function, i.e., an unnamed function that was defined at the point of use. The first example of the creation of a named function is in Subsection 4.1.5.

pp.26-28, Section 1.11

Additional Exercise

11. From the data frame `austpop`, extract a data frame that excludes the information for `ACT` and `NT`.

Chapter 2

p.39, Figures 2.10 and 2.11

Figures 2.10 and 2.11, which were intended to appear in color, appeared in black and white in the first printing. Figures 2.10 and 2.11 will need to be redrawn in color, or with a different choice of symbols for different levels of `target` (Figure 2.10) or different levels of `tint` (Figure 2.11), for the comments that follow to make visual sense.

p.40, lines 11-12

In order to get automatic labeling, specify:

```
xyplot(csoa ~ it | sex*agegp, data=tinting,
       groups=target, auto.key=list(columns=2))
```

p.40, footnotes 14 and 15

Better alternatives to the code in this footnotes are:

```
# Modified version of Figure 2.10
trellis.par.set(superpose.symbol=list(col=c("gray","black"),
                pch=c(16,15)))
xyplot(csoa ~ it|sex*agegp, data=tinting,
       groups=target, col=colr, pch=plotchar,
       auto.key=list(columns=2))

# Modified version of Figure 2.11
trellis.par.set(superpose.symbol=list(col=c("skyblue1",
                                            "skyblue4")[c(2,1,2)],
                                      pch=c(1,16,16)))
xyplot(csoa~it|sex*agegp, groups=tint, data=tinting,
       col=colr, pch=plotchar, type=c("p","smooth"), span=0.8,
       auto.key=list(columns=3))
 # The parameter "span" controls the extent of smoothing.
```

Comparisons between the four panels of these graphs are comparisons between different individuals. Each person is represented seven times in each panel, once for each combination of `tint` with `target` other than no tinting with high contrast target, which occurs twice. Thus the graphs exaggerate the strength of the evidence for differences between panels. Comparisons between different levels of tinting, or between the two contrasts, are made within individuals, and the graphs do not exaggerate the evidence for these differences. The problem is inherent in graphs that present data where there is more than one level of variation. Proper interpretation of the graph must have regard to the structure of variation within the data. There is further discussion below, in connection with the analysis of these data on pp.239-242.

pp.50-51, Section 2.6 – Exercises

Exercise 1

As worded in the first printing, this has some of the elements of a trick question!

The second graph demonstrates an unsatisfactory and misleading choice of breaks. This is most easily seen by tabulating the frequencies for the two graphs, thus:

```
table(cut(possum$age, breaks=0:9))   # For comparison
table(cut(possum$age, breaks=c(0,1.5,3,4.5,6,7.5,9)))
```

For the second graph, the ages fall into groups 1, (2,3), 4, (5,6), 7, and (8,9). The category (0, 1.5] catches just 1 year olds, the category (1.5, 3] catches ages 2 and 3, and so on. Thus the histogram that has breaks at intervals of 1.5 years is misleading.

Exercise 2

A problem with the simple form of density plot is that it has a non-zero density for ages less than one. This can be fixed by changing the code to, e.g.

```
plot(density(possum$age, na.rm=T, from=0.5), main="")
```

Chapter 3

Subsection 3.3: The use of random numbers

An important application, to the simulation of data that follow a presumed model, is not mentioned.

One way to check the distribution that was assumed for the random part of the model is to repeatedly simulate data values from that distribution, for comparison with the distribution of residuals obtained from fitting the model to the actual data. This approach will be demonstrated in Section 3.4.

More generally, data that follow the model may be repeatedly simulated, fitting the mode to each set of simulated data. The fits to the simulated data should yield model parameters that are comparable with those that were fitted to the actual data. Repeated fits to simulated data may be used to get an indication of the sampling variability in model parameters. This can be especially useful in cases where the theoretical distributions are not known.

Chapter 4

Subsection 4.4.1

The calculation of residuals can be done more directly, thus:

```
## Calculate standardized ("Pearson") residuals
x2 <- chisq.test(rareplants)
round(residuals(x2), 3)   # Pearson residuals are the default
```

Analysis of variance – Section 4.5

This is the first of three places in the book that discusses the analysis of data from designed experiments. Section 7.1 is designed to give insight into technical aspects of the handling of analysis of variance calculations. Sections 9.3 and 9.4 discuss models where there is more than one level of variation.

pp.103-106, Section 4.12 – Exercises

Exercise 3

Add: (i) If the null hypothesis is true, from what distribution are these p-values drawn. Plot the ordered p-values against the quantiles of the relevant distribution, perhaps using `qqplot()`.

(ii) *Modify the function so that each random value y in the sample of 10 is replaced by exp(y − 1). Repeat the calculation of 50 independent p-values. (Use `t.test()`, even though the null distribution will now be a little different from the t-distribution.) Plot the ordered t-statistics against the quantiles of the same reference distribution as in (i). Repeat this several times. Is there any indication that the p-values may not have the assumed null distribution?

(iii) Repeat (ii), but now calculating and plotting 1000 independent p-values.

Exercise 5

Add: *Repeat this calculation a number of times, comparing the variances that are obtained with $2/n$, where $n = 50$. Is the variance, on average, bigger or smaller than $2/n$.
(The correct comparison is with $\frac{2(n-1)}{n^2}$. There is a correlation of 0.5 between successive pairs of values of `y1`.)

Exercise 10

Add: Repeat the creation of the overlaid density plots, but now replacing the sample values by random values from a Normal distribution. Repeat this a number of times. Do the density plots from the elastic band data lie within the range of variation observed for the simulated data sets?

Exercise 12

Add: Perform an analysis of variance calculation that will check whether the height differences seem to be different for different pots.

Chapter 5

Subsection 5.1.3, page 111

```
## Most simply, use the function qreference() (DAAG package)
qreference(residuals(roller.lm), nrep=8, nrows=2)
```

Subsection 5.3.3, pp.118-119

The line in the left panel of Figure 5.9 is for the data set `elastic2`, while that in the right panel is for the data set `elastic1`. The following code gives a figure that is very similar to Figure 5.9:

```
panelci<-function(x, y)
{
    elastic.lm <- lm(y~x)
    pred <- predict(elastic.lm, interval="confidence")
    ord<-order(x)
    llines(x[ord], pred[ord, "fit"], lty=1, lwd=2)
    llines(lowess(x[ord], pred[ord, "upr"]), lty=2, lwd=2, col="grey")
    llines(lowess(x[ord], pred[ord, "lwr"]), lty=2, lwd=2, col="grey")
}
library(DAAG)
if(!exists("elastic2"))data(elastic2)
if(!exists("elastic1"))data(elastic1)
elastic <- rbind(elastic2,elastic1)
trial <- rep(c("Range of stretch 30-60 mm","Range of stretch 42-54 mm"),
           c(dim(elastic2)[1],dim(elastic1)[1]))
elastic$trial <- trial
xyplot(distance ~ stretch | trial, data=elastic,
       xlab="Amount of stretch (mm)", ylab="Distance moved (cm)",
       panel=panelci, aspect=1)
```

p.132, Section 5.11, Exercises

Exercise 4

To convert degrees Celsius to degrees kelvin, add 273.15 $\simeq$ 273. A logarithmic transformation is too extreme if `pressure` is not also transformed. A good starting point is the Clausius-Clapeyron equation that relates vapor pressure to temperature. Look up information on this equation on the web.

In the list of corrections, we have reworded this question.

Chapter 6

Analysis of the `hills` data; Section 6.3.3, pp.147-152

A final check should be to examine the component plus residual plots, as in Figure 6.6. Here is suitable code:

```
hills.lm <- lm(log(time) ~ log(climb)+log(dist), data=hills[-18,])
b1 <- hills.lm$coef[2]
b2 <- hills.lm$coef[3]
plot(log(hills$climb[-18]),
     b1*log(hills$climb[-18])+residuals(hills.lm))
panel.smooth(log(hills$climb[-18]),
             b1*log(hills$climb[-18])+residuals(hills.lm))
plot(log(hills$dist[-18]),
     b2*log(hills$dist[-18])+residuals(hills.lm))
panel.smooth(log(hills$dist[-18]),
             b2*log(hills$dist[-18])+residuals(hills.lm))
```

There is noticeable curvature in both the plots. This can be accommodated using the regression spline methodology of chapter 7.

An alternative to the use of component plus residual plots is the use of plots of partial residuals, using `termplot()`:

```
hills.lm <- lm(log(time) ~ log(climb)+log(dist), data=hills[-18,])
termplot(loghills.lm, partial=T, smooth=panel.smooth, col.term=0)
```

Use of `col.term=0` suppresses the line that otherwise appears.

Section 6.6 – Measures for the Comparison of Regression Models

Miller (2002) discusses a number of other measures, additional to those that we consider, with relevance to variable selection.

Section 6.6 – Problems with Many Explanatory Variables

Miller (2002) has extensive discussion of the issues that we consider, with attention to such theoretical results as are available. We have largely bypassed that discussion, moving directly to the use of resampling methods (cross-validation and the boostrap) to assess predictive accuracy. A weakness of Miller's account, from our perspective, is that it does not have regard to the frequent demand to transform variables prior to entering them into the regression equation. In this connection, we have relied heavily on ideas that are presented in Cook and Weisberg (1999), which Miller does not discuss. Variable selection cannot be totally separated from model selection.

The Use of Cross-Validation in Model Selection

We mention the use of cross-validation in variable selection. An issue here is the choice of number of folds. Hastie et al (2001) suggest five-fold or ten-fold cross-validation, as a good compromise. Leave-one-out cross-validation, where there are as many folds as there are data values, gives estimates that are unbiased when $n-1$ of the n data values are used for estimation, i.e., the estimate of accuracy is a slight under-estimate. The amount of the under-estimate increases if there are a smaller number of folds and fewer than $n-1$ of the data values are therefore used, in each fold, for deriving the estimates that are specific to that fold.

In the discussion of tree-based regression in chapter 10, cross-validation will determine the number of splits.

References for Further Reading

Miller, A.J. 2002. Subset Selection in Regression. Chapman & Hall/CRC, 2nd edn.

pp.172-174, Section 6.10: Exercises

Exercise 9

In the list of corrections, we suggest omission of the final 3+ lines, i.e. of
"Are there any advantages . . . the diagnostic plots."
(It adds little to what has gone before.)

Additional Exercise

10. Values in the data frame `carprice` can be calculated thus:

```
data(Cars93)
US <- Cars93$Origin=="USA"
carprice <- Cars93[US,  c("Type", "Min.Price","Price","Max.Price")]
carprice$Range.Price <- carprice$Max.Price - carprice$Min.Price
carprice$Range.Price <- round(carprice$Range.Price, 1)
carprice$RoughRange <- carprice$Range.Price+rnorm(48, sd=0.01)
carprice$RoughRange <- round(carprice$RoughRange, 2)
avMPG <- apply(Cars93[US, c("MPG.city","MPG.highway")], 1, mean)
carprice$gpm100 <- round(100/avMPG, 1)
```

Examine how the coefficients in the model objects `carprice1.lm`, `carprice2.lm` and `carprice.lm` change when

(i) Values of `RoughRange` are recalculated. Do this several times.

(ii) `Range.Price` and `gpm100` are each rounded to two decimal places, rather than to one decimal place.

Chapter 9

Note: Data frames in the nlme package

Note that factors that appear in data frames that are in the nlme package are ordered. A consequence is that, for use of the `relevel()` function, they must first be turned into unordered factors. For example

```
library(nlme); data(ergoStool)
ergoStool$Subject <- factor(ergoStool$Subject, ordered=FALSE)
ergoStool$Subject <- relevel(ergoStool$Subject, ref="1")
```

pp.239-242: Analysis of the `tinting` data

Following on from the comments in the final two lines on p.240, note that none of the main effects and interactions involving `agegp` and `sex` are significant at the conventional 5% level, though `agegp` comes close. This seems inconsistent with Figures 2.10 and 2.11, where it is the older males who seem to have the longer times. On the other hand, the interaction terms (`tint.L:agegpsenior`, `targethicon:agegpsenior`, `tint.L:targethicon` and `tint.L:sexm`) that are statistically significant stand out much less clearly in Figures 2.10 and 2.11.

To understand the reason for this, it may help to look at the relative amounts of evidence for the two different sets of comparisons, and at the consequences for the standard errors in the table of computer output on p.241.

- **Numbers of individuals:**

  ```
  > attach(tinting)
  > uid <- unique(tinting$id)
  > subs <- match(uid, tinting$id)
  > table(sex[subs], agegp[subs])

      younger older
    f 9       4
    m 4       9
  ```

  Standard errors in the computer output on p.241, for comparisons made at this level, are in the range 0.23 - 0.32.

- **Numbers of comparisons between levels of `tint` or `target`:** Each of these comparisons is made at least as many times as there are individuals, i.e., at least 26 times. Standard errors in the computer output on p.241, for comparisons made at this level, are in the range 0.042 - 0.058.

p. 245, Subsection 9.5.3, The general AR(p) model

Note that, as defined here, σ^2 is the variance of the independent and identically distributed innovations ϵ_t. For the Lake Huron data under the AR(1) model, $\widehat{\sigma^2}$ = `LH.mle$var.pred` = 0.509, while under the AR(2) model `LH.mle$var.pred` = 0.479. Contrast these with `var(LakeHuron)` = 1.738.

Additional Note

It may be shown that, for an AR(1) process

$$\mathrm{var}[\mathrm{X_t}] = \sigma^2/(1-\alpha^2)$$

Note also that, for an AR(1) process

$$E[\frac{\sum(X_i - \bar{X})^2}{n-1}] \simeq \frac{n-1-\alpha}{n-1} \simeq \sigma^2$$

Further Reading

The range of time series abilities that are available in R is extensive. Depending on what is required, it may now be the system of choice for time series analysis. Apart from the *ts* package and the `gls()` function in *nlme* see: *pear*, for periodic autoregressive models, including methods for plotting periodic time series; *fracdiff*, for fractionally differenced ARIMA "long memory" processes, where the correlation between time points decays very slowly as the points move apart in time; *strucchange*, for estimating and testing for change points; and the bundle of packages *dse1*, *dse2*, *dsepadi*, *syskern*, and *tframe*, for multivariate ARMA, for state space modeling, and associated forecasting.

As our discussion has indicated, ARIMA processes provide an integrated framework that include AR and ARMA processes as special cases. Their use in time series modeling and forecasting was popularized by Box and Jenkins in the late 1960s; hence they are often called Box-Jenkins models.

State space modeling approaches, which are now coming into wide use, are an important alternative. They provide a theoretical basis for the widely popular exponential forecasting methodology, and for various extensions of exponential forecasting. See Ord et al (1997) and Hyndman et al (2002). These approaches are intuitively appealing because they discount information from the remote past. Hyndman et al give an interesting and insightful comparison of a number of different forecasting approaches, in which methods of this type do well. Methods of this type are implemented in the `StructTS()` and `HoltWinters()` functions in the *ts* package.

References for Further Reading

Time Series

Hyndman, R.J., Koehler, A.B., Snyder, R.D. and Grose, S. 2002. A state space framework for automatic forecasting using exponential smoothing methods. International Journal of Forecasting 18: 439-454.

Ord, J.K., Koehler, A.B., Snyder, R.D. 1997. Estimation and prediction for a class of dynamic nonlinear statistical models. Journal of the American Statistical Association 92: 1621-1629.

p.258, Section 9.10 – Exercises

Exercise 3

Assume for the purposes of this calculation that σ is known. See the list of corrections.

The following results are relevant to the more usual case where σ is unknown:

- As defined on p.244, σ^2 is the variance of the innovation ϵ_t, not the variance of the observations X_t. [The variance of X_t is $\frac{\sigma^2}{1-\alpha^2}$.]
- The statistic $\frac{\sum(X_t-\bar{X})^2}{n-1-\alpha}$ ($\simeq \frac{\sum(X_t-\bar{X})^2}{n-1}$ providing that $n >> 1+\alpha$) is an almost unbiased estimate of var[$\mathrm{X_t}$].]

Additional Exercises

9.* Use the `arima.sim` function in the `ts` library to simulate 100000 values from an AR(1) process with $\alpha = -.5$. Now break this up in to 1000 series of length 100. If `x` is the series, a straightforward way to do this is to set

```
xx <- matrix(x, ncol=1000}
```

Now use the function `apply()` (Section 12.7.2) to find the means for each of these series of length 100. Compare

$$\frac{\sum(X_t-\bar{X})^2}{n-1}$$

with var[$\bar{X}$] estimated from the formula $\frac{\sigma^2}{n(1-\alpha)}$

For comparison, check the effect of using $\frac{\text{var[X]}}{n}$ to estimate the variance. First calculate the ordinary sample variance for each of our 1000 series. Then compute the average of these variance estimates and divide by the sample size, 100. (This gives a value that is also close to that predicted by the theory, roughly three times larger than the value that was obtained using the correct formula.)

10. Sugar content in cereal is monitored in two ways: a lengthy lab analysis and by using quick, inexpensive high performance liquid chromatography. The data in `frostedflakes` (*DAAG*)come from 101 daily samples of measurements taken using the two methods.
    (i) Obtain a vector of differences between the pairs of measurements.
    (ii) Plot the sample autocorrelation function of the vector of differences. Would an MA(1) model be more realistic than independence?
    (iii) Compute a confidence interval for the mean difference under the independence assumption and under the MA(1) assumption.

Chapter 11

Section 11.4: Propensity Scores in Regression Comparisons – Labor Training Data

There is a large element of arbitrariness in the extent of overlap demanded between the two distributions. Sensitivity analysis, checking how robust results are to the choices of cut-off point at the different stages of the analysis, is essential. When the sensitivity of the result to the choice of cut-off point is tested, it becomes apparent that the results are highly unstable. This is true whether regression methods are used directly, or the score (propensity) method is used. The following code, which appears in the second edition, uses the function `multilap()` to control the choice of cut-off point.

```
library(splines)
A.lm <- lm(formula = log(re78 + 47) ~ trt + (ns(age, 4) + educ +
           log(re74 + 47) + log(re75 + 47)) + I(re74 == 0) +
           I(re75 == 0) + (black + hisp + marr + nodeg),
           data = nsw74psid1)
summary(A.lm)$coef
## Footnote code
"multilap" <-
  function(df=nsw74psid1, maxf=20,
           colnames=c("educ","age","re74","re75","re78")){
    if(maxf==Inf)
      return(rep(TRUE, dim(df)[1]))
    if(length(maxf)==1)maxf <- c(1/maxf, maxf)
    trt <- df$trt
    common <- rep(TRUE, length(trt))
    for(vname in colnames){
      y0 <- df[trt==0,vname]
      y1 <- df[trt==1,vname]
      xchop <- overlapDensity(y0, y1, ratio=maxf,
                              compare.numbers=FALSE, plotit=FALSE)
      common <- common & df[,vname]>=xchop[1] & df[,vname]<=xchop[2]
}
invisible(common)
}
## Now run the function
common <- multilap(maxf=30)
```

```
xyplot(resid(A.lm) ~ fitted(A.lm), groups=nsw74psid1$trt[common],
       type=c("p","smooth"))

## ss 13.2.2: The use of propensity scores
common <- multilap(maxf=30)  # Mild preliminary filtering
## Calculate propensity scores: data frame nsw74psidA (DAAG)
disc.glm <- glm(formula = trt ~ age + educ + black + hisp + marr +
                nodeg + re74 + re75, family = binomial,
                data = nsw74psid1, subset=common)
Pscores <- predict(disc.glm)
nsw74psidB <- subset(nsw74psid1, common)
trt <-  nsw74psidB$trt
xchop <- overlapDensity(Pscores[trt==0], Pscores[trt==1],
                        compare.numbers=FALSE,
                        ratio=c(1/30, 30))
overlap <- Pscores > xchop[1] & Pscores < xchop[2]
with(subset(nsw74psidB, overlap), table(re78>0, trt))
AS.lm <- lm(log(re78+47) ~ trt + Pscores, data=nsw74psidB,
              subset =  overlap)
summary(AS.lm)$coef
AS.glm <- glm(I(re78>0) ~ trt + Pscores, data=nsw74psidB,
              subset =  overlap, family=binomial)
summary(AS.glm)$coef
## Use lm(), now limiting attention to data where re78>0
AS.lm <- lm(log(re78+47) ~ trt + Pscores, data=nsw74psidB,
            subset =  overlap&nsw74psidB$re78>0)
summary(AS.lm)$coef
```

The Dehejia and Wahba (1999) paper may be too optimistic in its assessment of what propensity score methods can achieve with these data. For example, try

```
## The next line has been changed
common <- multilap(maxf=25)  # NB: change from maxf=30
## Calculate propensity scores: data frame nsw74psidA (DAAG)
disc.glm <- glm(formula = trt ~ age + educ + black + hisp + marr +
                nodeg + re74 + re75, family = binomial,
                data = nsw74psid1, subset=common)
Pscores <- predict(disc.glm)
nsw74psidB <- subset(nsw74psid1, common)
trt <-  nsw74psidB$trt
## The next statement has been changed
xchop <- overlapDensity(Pscores[trt==0], Pscores[trt==1],
                        compare.numbers=FALSE,
                        ratio=c(1/25, 25)) # NB: change to c(1/25,25)
overlap <- Pscores > xchop[1] & Pscores < xchop[2]
with(subset(nsw74psidB, overlap), table(re78>0, trt))
AS.lm <- lm(log(re78+47) ~ trt + Pscores, data=nsw74psidB,
              subset =  overlap)
summary(AS.lm)$coef
```

The coefficients from this second set of calculations are notably different, and would suggest a different conclusion.

The advantage of the propensity score method is that it allows a simpler and more consistent rationale for the eventual choice of observations to exclude, and that it greatly improves the effectiveness of diagnostic plots as tools for checking the adequacy of the model. It was however necessary to screen observations before proceeding to the calculation that determined the propensity scores. Unless explanatory variables have approximately linear relationships, and the "true" propensities are a linear function of the scores, the methodology cannot be expected to work well.

Other methods – Support Vector Machines

Support vector machines have recently come into prominence. The *e1071* package in R handles the fitting of Support Vector Machines. Meyer et al (2003) use a number of different data sets to compare Support Vector Machines with other approaches.

Section 11.5: Further Reading

Computationally, principal components analysis uses the a principal components form of matrix decomposition, applied to a variance-covariance or correlation matrix. The variance-covariance matrix has the form $X'X$, where X is derived from the observations by variables matrix by subtracting off means in each column. For calculations with the correlation matrix, values in each column must, additionally, be scaled so that the squared values sum to 1.0.

Equivalently, it is possible to work directly with a version of X, and apply the singular value decomposition. This leads to a different form of description of the analysis. As an example, see Booth et al (2002), where the singular value decomposition is applied to a matrix that gives, for each of a number of age categories, mortalities for each of the years 1907-1999 or (for some of the analyses) 1968-1999. In the "Lee-Carter" methodology that is described and critically evaluated in that paper, scores on what is really the first principal component are used, after an adjustment that better reproduces the total number of deaths in any year, as a basis for mortality projections into the future.

Functional data analysis is an extension of such an approach.

References for further reading

Booth, H., Maindonald, J., and Smith, L. 2002. Applying Lee-Carter under conditions of variable mortality decline. Population Studies, 56: 325-336.

Meyer, D., Leisch, F., and Hornik, K. 2003. The Support Vector Machine under test. Neurocomputing 55: 169-186

pp.298-299, Section 11.6 – Exercises

For these exercises, be sure to use the corrected version of the function `overlap.density()`, which first appeared in version 0.34 of *DAAG*. The former argument `frac` has now become, more accurately, `ratio`.

Exercise 1

For each principal component in turn (i = 1, 2, 3 are as many as are likely to be useful), the comparison can be made graphically. For example, starting with the first principal component, plot the loadings for females against the loadings for all data combined, and plot the loadings for males against the loadings for all data combined, on the sample graph. Alternatively, the loadings for males can be plotted against the loadings for females.

Additional Exercise – continuation of Exercise 1

It would be useful to know whether differences that are observed in Exercise 1 are greater than would be expected as a result of sampling variation. A good way to check this is to take repeated bootstrap samples of the observations, and repeat the analysis for each such sample. A relevant bootstrap sample can be obtained by taking, with replacement, a sample whose size is the number of rows of observations.

Exercise 2

Note the correction. For *principal component* read, on each occurrence, *discriminant function.*

Chapter 12

Section 12.1: colors

The *RColorBrewer* package offers palettes are suitable for use where colors are used to distinguish different areas of space or different points in a graph. After installing the package, type in

```
library(RColorBrewer)
display.brewer.all()
display.brewer.all(name="div")  # divergent palettes
display.brewer.all("qual")      # qualitative palettes
display.brewer.all("seq")       # sequential palettes
```

A rationale for the choice of colors in the palettes, examples of their use, and references, may be found on the web site http://colorbrewer.org from which the palettes have (with permission) been taken.

Subsection 12.3.1: *`read.table()`*

Note also the use of the `quiet` and `skip` arguments. The `quiet` argument is, by default, set to `FALSE`, in which case, `scan()` prints a line showing how many items of data have been read. The `skip` argument tells `scan()` how many lines of the input file to omit before beginning reading in items.

p.310, Subsection 12.2.3: Functions for working with dates

In version 1.9.0, the *date* package has been superseded by functions for working with dates that are in R *base*. See `help(Dates)`, `help(as.Date)` and `help(format.Date)` for information. The document "Changes with R-1.9.0 and R-2.0.0" has a brief summary.

pp.324-327, Section 12.7: Matrices and Arrays

Note the following matrix operations:

```
X + Y         # Element-wise addition (X & Y both n by m)
X * Y         # Element-wise multiplication
X %*% B       # Matrix multiplication (X is n by k; B is k by m)
solve(X, Y)   # Solve XB = Y
```

Computational Speed

Here is a comparison of matrix computations with the equivalent computations with data frames.

```
> xy <- matrix(rnorm(500000),ncol=50)
> dim(xy)
[1] 10000    50
> system.time(xy+1)
[1] 0.05 0.00 0.08 0.00 0.00
> xy.df <- data.frame(xy)
> system.time(xy.df+1)
[1] 3.24 0.00 8.23 0.00 0.00
```

The two non-zero numbers are of interest. The first is processor time and the second is elapsed time. The above was on a 4Mhz Macintosh G4 laptop with 768MB of random access memory.

For a comparison of R 1.9.0, Matlab 6.2, S-PLUS 6.1, O-matrix 5.6, Scilab 2.7, Octave 2.1.42 and Ox 3.30, go to
http://www.sciviews.org/other/benchmark.htm

pp.330-334, Sections 12.9 & 12.10

The Use of a List to Pass Parameter Values

The following are equivalent:

```
mean(rnorm(20))
do.call("mean", args=list(x=rnorm(20)))
```

Use of `do.call()` allows the parameter list to be set up in advance of the call. The following function demonstrates a use for this.

```
simple.simulate.distribution <-
function(distn="weibull", parms=NULL, n=100){
## Simulates one of the distributions: weibull, gaussian,
## logistic, exponential
if(is.null(parms))
parms <- switch(distn, weibull=, list(shape=1), NULL)
## weibull requires a default shape parameter.
## ===================================================
## Choose the function that will generate the random sample
rfun <- switch(distn,
               weibull= rweibull,
               gaussian= rnorm,
               logistic= rlogis,
               exponential= rexp)

## Call the function. Use of do.call() allows giving
## the parameter list as a list of named values.
## (If the list is null, no parameters are passed.)
x <- do.call("rfun", args=c(parms, list(n=n)))

## Notice the use of c() to add another list item
invisible(x)
}
```

Note also `call()`, which sets up an unevaluated expression. The expression can be evaluated at some later time, using `eval()`.

```
> mean.call <- call("mean", x=rnorm(5))
> eval(mean.call)
```

```
[1] -0.6276536
> eval(mean.call)
[1] -0.6276536
```

Notice that the argument `x` was evaluated when `call()` was evoked. Hence the result is unchanged upon repeating the use of `eval()`. This can be verified by printing out the expression:

```
> mean.call
mean(x = c(-0.68467334794551, -0.376091734366091, -0.289459988631994,

-3.04694266628697, 1.25889972957396))
```

Environments

Every call to a function creates a frame that contains the local variables created in the function. This combines with the environment in which the function was defined to create a new environment.

The global environment, `.Globalenv`, is the workspace. This is frame number 0. The frame number increases by 1 with each new function call. Additionally, frames may be referred to by name. Use

`sys.nframe()` to get the number of the current evaluation frame

`sys.frame(sys.nframe())` to identify the frame by name

`sys.parent()` to get the number of the parent frame.

There are many other such functions, but these will do for now!

Here is a function that determines, from within the function, its name:

```
> test <- function(){
fname <- as.character(sys.call(sys.parent())[[1]])
fname
}
> test()
[1] "test"
> newtest <- test
> newtest()
[1] "newtest"
```

I like to call my figures, in the directory that relates to a paper that I am writing, `fig1()`, `fig2()`, etc. These functions will in turn call a function `gf()` that calls the graphics device, and specifies the file that will be used to store the graph. The definition of `gf()` is:

```
gf <-
    function(width=2.25, height=2.25, color=F, pointsize=8){
        funtxt <- sys.call(1)  # Sea
        fnam <- paste(funtxt, ".pdf", sep="")
        print(paste("Output will be to the file '",
              fnam, "'", sep=""))
        pdf(file=fnam, width=width, height=height, pointsize=
            pointsize)
    }
```

Now create a function that calls `gf()`:

```
graph1 <- function(){
    gf()             # Call with default parameters
    curve(sin, -pi, 2*pi)
    dev.off()
}
```

Output goes to the file **graph1.pdf**. For a function `graph2()` that calls `gf()`, the file name will be **graph2.pdf**. Similarly for `fig1()` or `fig2()` or any other function that calls `gf()`.

Bound and Unbound Variables

Variables that are passed as formal arguments to a function are bound variables. Local variables are those that are created within the body of the function. Unbound variables are first searched for in the frame of the function, then in the parent frame, and so on. If they are not found in any of the frames, then they are sought in the search list.

pp.334-337, Section 12.12: Exercises

Exercise 8

The list of corrections suggests a rewording that makes this a more satisfactory question.

Exercise 11

Note the reworded version of this question that is in the list of corrections.

Section 12.12: Additional exercise

12. The `match()` function and the `%in%` operator offer alternative ways to extract, from the `cuckoos` data frame (*DAAG*), rows that relate to wrens and robins. Demonstrate these alternatives.