Math3346 – Laboratory Exercises 1 and 2

Exercises for Use in Practicing R Skills

(the final page includes information on files on the DVD)

John Maindonald

July 21, 2008

Be selective: Rather than working through all exercises in detail (there will not be anything like enough time!), it will be preferable to work through the first few, then make a selection from the remainder. Asterisked exercises (especially Exercises 10-14 in Part II) are intended for those who want to be challenged, or to extend their hoRizons. The subdirectory **scripts** on the DVD includes the R script files **r-exs1.R** and **r-exs2.R**.

There is extensive use of datasets from the *DAAG* and (in Part II) *DAAGxtras* packages.

Contents

Part I
R Basics

1 Data Input

Exercise 1
The file **fuel.txt** is one of several files that the function `datafile()` (from *DAAG*), when called with a suitable argument, has been designed to place in the working directory. On the R command line, type `library(DAAG)`, then `datafile("fuel")`, thus:[a]

```
> library(DAAG)
> datafile(file = "fuel")
```

Alternatively, copy **fuel.txt** from the directory **data** on the DVD to the working directory.

Use `file.show()` to examine the file.[b] Check carefully whether there is a header line. Use the R Commander menu to input the data into R, with the name `fuel`. Then, as an alternative, use `read.table()` directly. (If necessary use the code generated by the R Commander as a crib.) In each case, display the data frame and check that data have been input correctly.

Note: If the file is elsewhere than in the working directory a fully specified file name, including the path, is necessary. For example, to input **travelbooks.txt** from the directory **data** on drive **D:**, type

```
> travelbooks <- read.table("D:/data/travelbooks.txt")
```

For input to R functions, forward slashes replace backslashes.

[a] This and other files used in these notes for practice in data input are also available from the web page `http://www.maths.anu.edu.au/~johnm/datasets/text/`.
[b] Alternatively, open the file in R's script editor (under Windows, go to File | Open script...), or in another editor.

Exercise 2
The files **molclock1.txt** and **molclock1.txt** are in the **data** directory on the DVD.[a]

As in Exercise 1, use the R Commander to input each of these, then using `read.table()` directly to achieve the same result. Check, in each case, that data have been input correctly.

[a] Again, these are among the files that you can use the function `datafile()` to place in the working directory.

2 The `paste()` Function

Exercise 3
Here are examples that illustrate the use of `paste()`:

```
> paste("Leo", "the", "lion")
> paste("a", "b")
> paste("a", "b", sep = "")
> paste(1:5)
> paste(1:5, collapse = "")
```

What are the respective effects of the parameters `sep` and `collapse`?

3 Missing Values

Exercise 4
The following counts, for each species, the number of missing values for the column `root` of the data frame `rainforest` (*DAAG*):

```
> library(DAAG)
> with(rainforest, table(complete.cases(root), species))
```

For each species, how many rows are "complete", i.e., have no values that are missing?

Exercise 5
For each column of the data frame `Pima.tr2` (*MASS*), determine the number of missing values.

4 Subsets of Dataframes

Exercise 6
Use `head()` to check the names of the columns, and the first few rows of data, in the data frame `rainforest` (*DAAG*). Use `table(rainforest$species)` to check the names and numbers of each species that are present in the data. The following extracts the rows for the species *Acmena smithii*

```
> library(DAAG)
> Acmena <- subset(rainforest, species == "Acmena smithii")
```

The following extracts the rows for the species `Acacia mabellae` and `Acmena smithii`

```
> AcSpecies <- subset(rainforest, species %in% c("Acacia mabellae",
+     "Acmena smithii"))
```

Now extract the rows for all species except `C. fraseri`.

Exercise 7
Extract the following subsets from the data frame `ais` (*DAAG*):

(a) Extract the data for the rowers.

(b) Extract the data for the rowers, the netballers and the tennis players.

(c) Extract the data for the female basketabllers and rowers.

5 Scatterplots

Exercise 8
Using the Acmena data from the data frame `rainforest`, plot `wood` (wood biomass) vs `dbh` (diameter at breast height), trying both untransformed scales and logarithmic scales. Here is suitable code:

```
> Acmena <- subset(rainforest, species == "Acmena smithii")
> plot(wood ~ dbh, data = Acmena)
> plot(wood ~ dbh, data = Acmena, log = "xy")
```

Exercise 8, continued
Use of the argument `log="xy"` gives logarithmic scales on both the x and y axes. For purposes of adding a line, or other additional features that use x and y coordinates, note that logarithms are to base 10.

```
> plot(wood ~ dbh, data = Acmena, log = "xy")
> Acmena10.lm <- lm(log10(wood) ~ log10(dbh), data = Acmena)
> abline(Acmena10.lm)

> coef(Acmena10.lm)
> Acmena.lm <- lm(log(wood) ~ log(dbh), data = Acmena)
> coef(Acmena.lm)
```

Write down the equation that gives the fitted relationship between `wood` and `dbh`.

Exercise 9
The `orings` data frame gives data on the damage that had occurred in US space shuttle launches prior to the disastrous Challenger launch of January 28, 1986. Only the observations in rows 1, 2, 4, 11, 13, and 18 were included in the pre-launch charts used in deciding whether to proceed with the launch. Add a new column to the data frame that identifies rows that were included in the pre-launch charts. Now make three plots of `Total` incidents against `Temperature`:

(a) Plot only the rows that were included in the pre-launch charts.

(b) Plot all rows.

(c) Plot all rows, using different symbols or colors to indicate whether or not points were included in the pre-launch charts.

Comment, for each of the first two graphs, whether and open or closed symbol is preferable. For the third graph, comment on the your reasons for choice of symbols.

Use the following to identify rows that hold the data that were presented in the pre-launch charts:

```
> included <- logical(23)
> included[c(1, 2, 4, 11, 13, 18)] <- TRUE
```

The construct `logical(23)` creates a vector of length 23 in which all values are `FALSE`. The following are two possibilities for the third plot; can you improve on these choices of symbols and/or colors?

```
> plot(Total ~ Temperature, data = orings, pch = included + 1)
> plot(Total ~ Temperature, data = orings, col = included + 1)
```

Exercise 10
Using the data frame `oddbooks`, use graphs to investigate the relationships between:

(a) weight and volume;

(b) density and volume;

(c) density and page area.

6 Factors

Exercise 11
Investigate the use of the functions `as.character()` and `unclass()` with a factor argument. Comment on their use in the following code:

```
> par(mfrow = c(1, 2), pty = "s")
> plot(weight ~ volume, pch = unclass(cover), data = allbacks)
> plot(weight ~ volume, data = allbacks, type = "n")
> with(allbacks, text(weight ~ volume, labels = as.character(cover)))
> par(mfrow = c(1, 1))
```

[`mfrow=c(1,2)`: plot layout is 1 row × 2 columns; `pty="s"`: square plotting region.]

Exercise 12
Run the following code:

```
> gender <- factor(c(rep("female", 91), rep("male", 92)))
> table(gender)
> gender <- factor(gender, levels = c("male", "female"))
> table(gender)
> gender <- factor(gender, levels = c("Male", "female"))
> table(gender)
> rm(gender)
```

The output from the final `table(gender)` is

```
gender
  Male female
     0     91
```

Explain the numbers that appear.

7 Stripcharts (base graphics) and Stripplots (*lattice*)

Exercise 13
Look up the help for the lattice function `dotplot()`. Compare the following, noting the differences in syntax between the lattice graphics function `stripplot()` and the base graphics function `stripchart()`.

```
> with(ant111b, stripchart(harvwt ~ site))
> library(lattice)
> stripplot(site ~ harvwt, data = ant111b)
> stripplot(harvwt ~ site, data = ant111b)
```

Exercise 14
Check the class of each of the columns of the data frame `cabbages` (*MASS*). Do side by side plots of `HeadWt` against `Date`, for each of the levels of `Cult`.

```
> stripplot(Date ~ HeadWt | Cult, data = cabbages)
```

The lattice graphics function `stripplot()` seems generally preferable to the base graphics function `stripchart()`. It has functionality that `stripchart()` lacks, and a consistent syntax that it shares with other lattice functions.

Exercise 15
In the data frame `nsw74psid3`, use `stripplot()` to compare, between levels of `trt`, the continuous variables `age`, `educ`, `re74` and `re75`
It is possible to generate all the plots at once, side by side. A simplified version of the plot is:

```
> stripplot(trt ~ age + educ, data = nsw74psid1, outer = T, scale = "free")
```

What are the effects of `scale = "free"`, and `outer = TRUE`? (Try leaving these at their defaults.)

8 Tabulation

Exercise 16
In the data set `nsw74psid3`, compare for each of the two levels of `trt`, the relative numbers for each of the factors `black`, `hisp` (hispanic), and `marr` (married).

9 Sorting

Exercise 17
Sort the rows in the data frame `Acmena` in order of increasing values of `dbh`.
[Hint: Use the function `order()`, applied to `age` to determine the order of row numbers required to sort rows in increasing order of age. Reorder rows of `Acmena` to appear in this order.]

```
> Acmena <- subset(rainforest, species == "Acmena smithii")
> ord <- order(Acmena$dbh)
> acm <- Acmena[ord, ]
```

Sort the row names of `possumsites` (*DAAG*) into alphanumeric order. Reorder the rows of `possumsites` in order of the row names.

10 For Loops

Exercise 18

(a) Create a `for` loop that, given a numeric vector, prints out one number per line, with its square and cube alongside.

(b) Look up `help(while)`. Show how to use a `while` loop to achieve the same result.

(c) Show how to achieve the same result without the use of an explicit loop.

11 A Function

Exercise 19
The following function calculates the mean and standard deviation of a numeric vector.

```
> meanANDsd <- function(x) {
+     av <- mean(x)
+     sdev <- sd(x)
+     c(mean = av, sd = sdev)
+ }
```

Modify the function so that: (a) the default is to use `rnorm()` to generate 20 random normal numbers, and return the standard deviation; (b) if there are missing values, the mean and standard deviation are calculated for the remaining values.

12 Further Practice with Data Input

*Exercise 20**
The function `read.csv()` is a variant of `read.table()` that is designed to read in comma delimited files such as may be obtained from Excel. Use this function to read in the file **crx.data** that is available from the web page `http://mlearn.ics.uci.edu/databases/credit-screening/`.
Check the file **crx.names** to see which columns should be numeric, which categorical and which logical. Make sure that the numbers of missing values in each column are the number given in the file **crx.names**

For a first pass at inputting the data, try:

```
> crxpage <- "http://mlearn.ics.uci.edu/databases/credit-screening/crx.data"
> crx <- read.csv(url(crxpage), header = TRUE)
```

*Exercise 21**
For a challenging data input task, input the data from **bostonc.txt**.[a]
Examine the contents of the initial lines of the file carefully before trying to read it in. It will be necessary to change `sep`, `comment.char` and `skip` from their defaults. Note that `\t` denotes a tab character.

[a] Use `datafile("bostonc")` to place it in the working directory, or access the copy on the DVD.

13 Data Input from a Web Page

*Exercise 22**
With a live internet connection, files can be read directly from a web page. Here are examples:

```
> webfolder <- "http://www.maths.anu.edu.au/~johnm/datasets/text/"
> webpage <- paste(webfolder, "molclock.txt", sep = "")
> molclock <- read.table(url(webpage))
```

At a time when a live internet connection is available, use this approach to input the file **travelbooks.txt** that is available from this same web page.

Part II
Practice with R

1 Information about the Columns of Data Frames

Exercise 1
Functions that may be used to get information about data frames include `str()`, `dim()`, `row.names()` and `names()`. Try each of these functions with the data frames `allbacks`, `ant111b` and `tinting` (all in *DAAG*).
For getting information about each column of a data frame, use `sapply()`. For example, the following applies the function `class()` to each column of the data frame `ant111b`.

```
> library(DAAG)
> sapply(ant111b, class)
```

For columns in the data frame `tinting` that are factors, use `table()` to tabulate the number of values for each level.

2 A Tabulation Exercise

Exercise 2
Tabulate the number of observations in each of the different districts in the data frame `rockArt` (*DAAGxtras*). Create a factor `groupDis` in which all `Districts` with less than 5 observations are grouped together into the category `other`.

```
> library(DAAGxtras)
> groupDis <- as.character(rockArt$District)
> tab <- table(rockArt$District)
> le4 <- rockArt$District %in% names(tab)[tab <= 4]
> groupDis[le4] <- "other"
> groupDis <- factor(groupDis)
```

3 A For Loop

Exercise 3
The following code uses a `for` loop to plot graphs that compare the relative population growth (here, by the use of a logarithmic scale) for the Australian states and territories.

```
> library(DAAG)
> oldpar <- par(mfrow = c(2, 4))
> for (i in 2:9) {
+     plot(austpop[, 1], log(austpop[, i]), xlab = "Year", ylab = names(austpop)[i],
+         pch = 16, ylim = c(0, 10))
+ }
> par(oldpar)
```

Which Australian adminstration(s) showed the most rapid increase in the early years? Which showed the most rapid increase in later years?

4 Data Exploration – Distributions of Data Values

Exercise 4
The data frame `rainforest` (*DAAG* package) has data on four different rainforest species. Use `table(rainforest$species)` to check the names and numbers of the species present. In the sequel, attention will be limited to the species Acmena *smithii*. The following plots a histogram showing the distribution of the diameter at base height:

```
> library(DAAG)
> Acmena <- subset(rainforest, species == "Acmena smithii")
> hist(Acmena$dbh)
```

Above, frequencies were used to label the the vertical axis (this is the default). An alternative is to use a density scale (`prob=TRUE`). The histogram is interpreted as a crude density plot. The density, which estimates the number of values per unit interval, changes in discrete jumps at the breakpoints (= class boundaries). The histogram can then be directly overlaid with a density plot, thus:

```
> hist(Acmena$dbh, prob = TRUE, xlim = c(0, 50))
> lines(density(Acmena$dbh, from = 0))
```

Why use the argument `from=0`? What is the effect of omitting it?

[Density estimates, as given by R's function `density()`, change smoothly and do not depend on an arbitrary choice of breakpoints, making them generally preferable to histograms. They do sometimes require tuning to give a sensible result. Note especially the parameter `bw`, which determines how the bandwidth is chosen, and hence affects the smoothness of the density estimate.]

5 Random Samples

Exercise 5
By taking repeated random samples from the normal distribution, and plotting the distribution for each such sample, one can get an idea of the effect of sampling variation on the sample distribution. A random sample of 100 values from a normal distribution (with mean 0 and standard deviation 1) can be obtained, and a histogram and overlaid density plot shown, thus:

```
> y <- rnorm(100)
> hist(y, probability = TRUE)
> lines(density(y))
```

(a) Take 5 samples of size 25, then showing the plots.

(b) Take 5 samples of size 100, then showing the plots.

(c) Take 5 samples of size 500, then showing the plots.

(d) Take 5 samples of size 2000, then showing the plots.

(Hint: By preceding the plots with `par(mfrow=c(4,5))`, all 20 plots can be displayed on the one graphics page. To bunch the graphs up more closely, make the further settings `par(mar=c(3.1,3.1,0.6,0.6), mgp=c(2.25,0.5,0))`)

Comment on the usefulness of a sample histogram and/or density plot for judging whether the population distribution is likely to be close to normal.

Histograms and density plots are, for "small" samples, notoriously variable under repeated sampling. This is true even for sample sizes as large as 50 or 100.

Exercise 6
This explores the function `sample()`, used to take a sample of values that are stored or enumerated in a vector. Samples may be with or without replacement; specify `replace = FALSE` (the default) or `replace = TRUE`. The parameter `size` determines the size of the sample. By default the sample has the same size (length) as the vector from which samples are taken. Take several samples of size 5 from the vector `1:5`, with `replace=FALSE`. Then repeat the exercise, this time with `replace=TRUE`. Note how the two sets of samples differ.

Exercise 7
If in Exercise 4 above a new random sample of trees could be taken, the histogram and density plot would change. How much might we expect them to change?

The boostrap approach treats the one available sample as a microcosm of the population. Repeated with replacement samples are taken from the one available sample. This is equivalent to repeating each sample value and infinite number of times, then taking random samples from the population that is thus created. The expectation is that variation between those samples will be comparable to variation between samples from the original population.

(a) Take repeated (5 or more) bootstrap samples from the Acmena dataset of Exercise 4, and show the density plots. [Use `sample(Acmena$dbh, replace=TRUE)`].

(b) Repeat, now with the `cerealsugar` data from *DAAG*.

6 Smooth Curves

*Exercise 8**
The following compares three different smoothing functions. Comment on the different syntax and, in the case of `lowess()`, the different default output that is returned. Why, for the smooth obtained using `lowess()`, is it necessary to sort data in order of values of `dbh`? (Try omitting the ordering, and observe the result.)

```
> Acmena <- subset(rainforest, species == "Acmena smithii")
> plot(wood ~ dbh, data = Acmena)
> ord <- order(Acmena$dbh)
> with(Acmena[ord, ], lines(predict(loess(wood ~ dbh)) ~ dbh))
> plot(wood ~ dbh, data = Acmena)
> with(Acmena, panel.smooth(dbh, wood))
```

For each of the functions just noted, what are the parameters that control the smoothness of the curve? What, in each case, is the default?

7 Information on Workspace Objects

Exercise 9
An R workspace includes objects `possum1`, `possum2`, ...`possum5`. The folowing shows how to get the size of one of these objects one at a time.

```
> possum1 <- rnorm(10)
> object.size(possum1)
```

The names of the objects can be obtained with

```
> nam <- ls(pattern = "^possum")
```

To get the sizes from the names that are held in `nam`, do

```
> sapply(nam, function(x) object.size(get(x)))
```

Create objects `possum2`, ...`possum5`, and enter this command. Explain the successive steps in the computation.
[Hint: Compare `class(possum1)` with `class("possum1")`, and `object.size(possum1)` with `object.size("possum1")`]

*Exercise 10**
The function `ls()` lists, by default, the names of objects in the current environment. If used from the command line, it lists the objects in the workspace. If used in a function, it lists the names of the function's local variables. To get a listing of the contents of the workspace, do the following

```
> workls <- function() ls(name = ".GlobalEnv")
> workls()
```

(a) If `ls(name=".GlobalEnv")` is replaced by `ls()`, the function lists the names of its local variables. Modify `workls()` so that you can use it to demonstrate this.
[Hint: Consider adapting `if(is.null(name))ls())` for the purpose.]

(b) Write a function that calculates the sizes of all objects in the workspace, then listing the names and sizes of the largest ten objects.

8 Different Ways to Do a Calculation – Timings

*Exercise 10**
This exercise will investigate the relative times for alternative ways to do a calculation. The function `system.time()` will provide timings. The numbers that are printed on the command line, if results are not assigned to an output object, are the user cpu time, the system cpu time, and the elapsed time.
First, create both matrix and data frame versions of a largish data set.

```
> xxMAT <- matrix(runif(480000), ncol = 50)
> xxDF <- as.data.frame(xxMAT)
```

Exercise 11, continued*
Repeat each of the calculations that follow several times, noting the extent of variation between repeats. If there is noticeable variation, make the setting `options(gcFirst=TRUE)`, and check whether this leads to more consistent timings.
NB: If your computer chokes on these calculations, reduce the dimensions of `xxMAT` and `xxDF`

(a) The following compares the times taken to increase each element by 1:

```
> system.time(invisible(xxMAT + 1))[1:3]
> system.time(invisible(xxDF + 1))[1:3]
```

(b) Now compare the following alternative ways to calculate the means of the 50 columns:

```
> system.time(av1 <- apply(xxMAT, 2, mean))[1:3]
> system.time(av1 <- sapply(xxDF, mean))[1:3]
> system.time({
+     av2 <- numeric(50)
+     for (i in 1:50) av[i] <- mean(xxMAT[, i])
+ })[1:3]
> system.time({
+     av2 <- numeric(50)
+     for (i in 1:50) av[i] <- mean(xxDF[, i])
+ })[1:3]
> system.time({
+     colOFones <- rep(1, dim(xxMAT)[2])
+     av3 <- xxMAT %*% colOFones/dim(xxMAT)[2]
+ })[1:3]
```

(c) Pick one of the above calculations. Vary the number of rows in the matrix, keeping the number of columns constant, and plot each of user CPU time and system CPU time against number of rows of data.

Why is matrix multiplication is so efficient, relative to equivalent calculations that use `apply()`, or that use for loops?

9 Functions – Making Sense of the Code

*Exercise 12**
Data in the data frame `fumig` (*DAAGxtras*) are from a series of trials in which produce was exposed to a fumigant over a 2-hour time period. Concentrations of fumigant were measured at times 5, 10, 30, 60, 90 and 120 minutes. Code given following this exercise calculates a concentration-time (c-t) product that measures exposure to the fumigant, leading to the measure `ctsum`.

Examine the code in the three alternative functions given below, and the data frame `fumig` (in the `DAAGxtras` package) that is given as the default argument for the parameter `df`. Do the following:

(a) Run all three functions, and check that they give the same result.

(b) Annotate the code for `calcCT1()` to explain what each line does.

(c) Are fumigant concentration measurements noticeably more variable at some times than at others?

(d) Which function is fastest? [In order to see much difference, it will be necessary to put the functions in loops that run perhaps 1000 or more times.]

Code for 3 functions that do equivalent calculations

```
> "calcCT1" <- function(df = fumig, times = c(5, 10, 30, 60, 90,
+     120), ctcols = 3:8) {
+     multiplier <- c(7.5, 12.5, 25, 30, 30, 15)
+     m <- dim(df)[1]
+     ctsum <- numeric(m)
+     for (i in 1:m) {
+         y <- unlist(df[i, ctcols])
+         ctsum[i] <- sum(multiplier * y)/60
+     }
+     df <- cbind(ctsum = ctsum, df[, -ctcols])
+     df
+ }
> "calcCT2" <- function(df = fumig, times = c(5, 10, 30, 60, 90,
+     120), ctcols = 3:8) {
+     multiplier <- c(7.5, 12.5, 25, 30, 30, 15)
+     mat <- as.matrix(df[, ctcols])
+     ctsum <- mat %*% multiplier/60
+     cbind(ctsum = ctsum, df[, -ctcols])
+ }
> "calcCT3" <- function(df = fumig, times = c(5, 10, 30, 60, 90,
+     120), ctcols = 3:8) {
+     multiplier <- c(7.5, 12.5, 25, 30, 30, 15)
+     mat <- as.matrix(df[, ctcols])
+     ctsum <- apply(mat, 1, function(x) sum(x * multiplier))/60
+     cbind(ctsum = ctsum, df[, -ctcols])
+ }
```

10 *Use of `sapply()` to Give Multiple Graphs

*Exercise 13**

Here is code for the calculations that compare the relative population growth rates for the Australian states and territories, but avoiding the use of a loop:

```
> oldpar <- par(mfrow = c(2, 4))
> invisible(sapply(2:9, function(i, df) plot(df[, 1], log(df[,
+     i]), xlab = "Year", ylab = names(df)[i], pch = 16, ylim = c(0,
+     10)), df = austpop))
> par(oldpar)
```

Run the code, and check that it does indeed give the same result as an explicit loop.
[Use of `invisible()` as a wrapper suppresses printed output that gives no useful information.]
Note that `lapply()` could be used in place of `sapply()`.

There are several subtleties here:

(i) The first argument to `sapply()` can be either a list (which is, technically, a non-atomic vector) or a vector.[1] Here, we have supplied the vector 2:9

[1] By "vector" we usually mean an atomic vector, with "atoms" that are of one of the modes "logical", "integer", "numeric", "complex", "character"' or "raw". (Vectors of mode "raw" can for our purposes be ignored.)

(ii) The second argument is a function. Here we have supplied an anonymous function that has two arguments. The argument `i` takes as its values, in turn, the sucessive elements in the first argument to `sapply`

(iii) Where as here the anonymous function has further arguments, they are supplied as additional arguments to `sapply()`. Hence the parameter `df=austpop`.

11 *The Internals of R – Functions are Pervasive

*Exercise 14**
This exercise peeks into the internals of R's handling arithmetic and related computations. Those internals are close enough to the surface that users can experiment with them.

The binary arithmetic operators `+`, `-`, `*`, `/` and `^` are implemented as functions. (R is a functional language; albeit with features that compromise its purity as a member of this genre!) Try the following:

```
> 2 + 5
> 10 - 3
> 2/5
> (5 + 2) * (3 - 7)
```

There are two other binary arithmetic operators – `%%` and `%/%`. Look up the relevant help page, and explain, with examples, what they do. Try

```
> (0:25)%/%5
> (0:25)%%5
```

Of course, these are also implemented as functions. Write code that demonstrates this.

Note also that `[` is implemented as a function. Try

```
> z <- c(2, 6, -3, NA, 14, 19)
> z[5]
> heights <- c(Andreas = 178, John = 185, Jeff = 183)
> heights[c("Jeff", "John")]
```

Rewrite these using the usual syntax.

Use this syntax to extract, from the data frame `possumsites` (*DAAG*), the altitudes for Byrangery and Conondale.

Note: Expressions in which arithmetic operators appear as explicit functions with binary arguments translate directly into postfix reverse Polish notation, introduced in 1920 by the Polish logician and mathematician Jan Lukasiewicz. Postfix notation is widely used in the interpreters and compilers that translate computer language code into machine or assembly language instructions. See the Wikipedia article "Reverse Polish Notation".

A Running R from the DVD, & List of Files

Running R from the DVD: For running R from the DVD, use as target the file **Rgui.exe** in the subdirectory **R-2.7.1/bin/**.[2]

Right click on the desktop, then click on New | Target, browse to find **Rgui.exe**, click on OK, click on Next, give the shortcut a suitable name, and click on Finish. Finally, right click on the icon, click on Properties, and set the target to the directory that you want as the working directory.

Accessing packages from the DVD: Packages that are in the R-2.7.1 installation on the DVD can be accessed by an R installation on the computer's hard drive. If the DVD is in drive **D:**, enter:

```
.libPaths("D:/R-2.7.1/library")
```

For execution whenever a session is started in a working directory, create a `.First()` function that includes this statement, for inclusion in the workspace image that is saved at the end of the session.

Contents of Directories

pdf: pdf files for exercises, notes, overheads, **Rsetups.pdf** that describes (among other matters) use of the Tinn-R editor under Windows, and **Sweave-info.pdf** that describes Sweave usage.

scripts: Text (.R) files of code from the exercises & the notes. The file r-exs3.R has code for some additional regression exercises.

data: Files **bestTimes.txt**, **bostonc.txt** (challenging), **molclock1.txt** and **travelbooks.txt** – for practice with data input.

RData: Image files holding datasets and functions – **bomdata.RData** (historical annual Australian climate data (regions and states), **bomregions.RData** (regions only), **housetemps.RData** (thermacron temperature traces, at a 20 minute resolution, at five locations in a house, over 27 successive days, plus three functions for plotting data). Files **edc-co2.RData** and **edc-temp.RData** are Epica Dome C ice core data (CO_2 and temperature). The file **hadcrugl.RData** has global monthly temperature anomalies, for January 1850 to April 2008.

Functions: Files **getbom.RData** (function for extracting climate data from an Australian Bureau of Meteorology web page), **bomfuns.RData** (extract such data for states or defined regions), **bomts.RData** (does time series calculations on weather data), **bomplots.RData** (plot data) and **breaklabels.RData** (elegant ploting of axis tick labels, for data created using cut(), when the lower limit is -infinity) or the upper limit is infinity).

packages and packages-add: Packages (binaries) for Windows.

R-2.7.1: This directory tree holds an R executable in which all packages in the **packages** subdirectory are installed. Instructions for running R from the DVD, or for accessing packages that are in the DVD installation, were given above.

win-binaries: Setup files for R-2.7.1 (**R-2.7.1-win32.exe**), the emacs editor and ESS (**emacs-22.2-modified-2.exe**), the Tinn-R editor (**Tinn-R_1.19.4.7_setup.exe**), GGobi (**ggobi-2.1.7.exe**), GTK2 (**gtk-dev-2.12.9-win32-1.exe**, required for running GGobi and for the *rattle* package), and OpenBUGS (**OpenBUGS.zip**, for Bayesian inference using Gibbs sampling). GTK2 interfaces to R via the *RGtk* package. Ggobi interfaces to R via the *rggobi* package. OpenBUGS interfaces via R to the *BRugs* package.

Sweave: See **Sweave-info.pdf**, in the directory **pdf**, for details.

[2] This is a copy of the directory tree from the installation on my laptop.