Appendix: the R statistical software package

Below we give only a very brief introduction to R, a language and environment for statistical computing and graphics. It is freely available for a range of platforms from

`http://cran.r-project.org/`

where you can also find extensive R documentation.

On-line help is available: to start with, try ?*help* at the command prompt > and more generally use ?xxx for help about xxx.

1 Objects

R is based around the idea of objects. Loosely these can be divided into data objects and functions, that is objects which act on data. Objects are manipulated via commands which may be assignments such as

$$a <- 1$$

which sets a equal to 1, or expressions such as

$$1 + exp(a)$$

which evaluates $1 + e^a$.

1.1 Data objects

There are many of these, but we shall mainly need the following

(a) *Vectors*: Created by the function c

$$\begin{aligned} x &<- c(2,3,4,5) \\ y &<- c("a","b","c","d") \end{aligned}$$

The first assignment is equivalent to

$$x <- 2{:}5$$

Vector elements can be easily accessed or modified by $y[1]$ (which returns "a") etc. Vector arithmetic works as expected: $x+1$ adds 1 to every element of x etc.

(b) *Factors*: A special kind of vector which allows functions to treat elements as levels of a categorical variable. Eg:

$$letters <- factor(c("a","b","c","d"))$$

Model statements (more later) often use the $as.factor()$ command to enable functions to treat vectors as factors.

(c) *Matrices:* Data are read by columns, so that the command

$$mat <- matrix(1{:}4, nrow = 2, ncol = 2)$$

creates the matrix

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Matrices can be built up from vectors row-wise (using *rbind*) or column-wise (*cbind*). For example

$$mat2 <- cbind(mat, c(7, 8))$$

creates the matrix

$$\begin{bmatrix} 1 & 3 & 7 \\ 2 & 4 & 8 \end{bmatrix}$$

Elements can be accessed using $mat[i, j]$, rows via $mat[i,]$ and columns via $mat[, j]$. For example $mat[1,]$ returns the first row. The usual operators are available, eg % * % is used for matrix multiplication.

(d) *Data frames:* These can be thought of as a list of variables of the same length, but possibly different types. Eg:

$y <- c(5.1, 4.2, 3.7, 8.0)$
$x1 <- factor(c("AB", "AB", "AA", "BB"))$
$x2 <- factor(c("male", "female", "male", "male")$
$test.data <- data.frame(response = y, genotype = x1, sex = x2)$
$test.data2 <- data.frame(y, x1)$

Variables can then be accessed using eg *test.data$response*.

1.2 Functions

Objects may be manipulated using the many R built-in functions. Use is intuitive, eg $sum(y)$ evaluates the sum of the elements of the vector y. They can be used in assignments in the obvious way, eg $ybar <- mean(y)$. The function *apply* can be used to apply a function to the rows or columns of a matrix, for example $apply(mat, 1, sum)$ returns 4 6 (row sums) and $apply(mat, 2, sum)$ returns 3 7 (column sums).

One of the advantages of R is the ease with which you can write your own functions. For example

$$f <- function(x = 3) \quad x^2 + 1$$

creates a function f. Subsequently the command $f(2)$ returns 5 and $f(1{:}3)$ returns 2 5 10, while $f()$ returns 10 because 3 has been assigned as the default value for x.

R is particularly useful for its graphics capabilities. The simplest plots can be obtained using *plot* with key arguments

$$plot(x, y, xlim = range(x), ylim = range(y), type = "p", main, xlab, ylab)$$

where *xlim* and *ylim* are vectors defining the axes limits, *type* is a character defining the type of plot, and the final three arguments are character strings defining axes and plot labels. Only the first two arguments are required. For example

$$x <-(0{:}100)/10$$
$$plot(x, f(x), type = "l", xlab = "x", ylab = "y")$$

creates a plot of the function f in the interval (0,10) with points evaluated at intervals of 0.1 on the x-axis and connected by straight lines. The functions *postscript* and *pdf* can be used to save plots as postscript or pdf files.

1.3 Model formulae

These provide a natural way of specifying linear models, and are used by many R functions. They are best introduced by examples:

$$ex1.lm <-lm(response \sim sex, data = test.data)$$
$$ex2.lm <-lm(time \sim distance + climb + distance : climb)$$
$$ex3.lm <-lm(time \sim distance * climb)$$

fits the models

$$\begin{aligned} response &= \beta_M I_{sex=male} + \beta_F I_{sex=female} + \varepsilon \\ time &= \beta_0 + \beta_1 distance + \beta_2 climb + \beta_3 (distance * climb) + \varepsilon \\ time &= \beta_0 + \beta_1 distance + \beta_2 climb + \beta_3 (distance * climb) + \varepsilon \end{aligned}$$

by least squares. Note the last two models are identical. The *data* option, if present, specifies the dataframe from which the variables should be extracted. If not present, *lm*() looks for variables of the appropriate name in the working directory. Note that *lm* returns an object of class *lm*. This can in turn be manipulated by other functions: try $plot(ex1.lm)$ and $plot.lm(ex1.lm)$ for example.

2 Fitting GLM in R

The key function is *glm*, with syntax similar to the *lm* function for fitting standard linear regression models. The key arguments are:

glm(formula, family, data,subset, control)

where:

- *formula* is a model formula just as for linear models, ie for a response y and explanatory variables $x1, x2$ we might have

  $$y \sim x1 + x2$$

  To add the interaction between $x1$ and $x2$, we would use

  $$y \sim x1 + x2 + x1 : x2 \text{ or, equivalently, } y \sim x1 * x2$$

  As usual, the intercept term is fitted automatically; to remove it, use eg

  $$y \sim -1 + x1 + x2$$

- *family* indicates the distribution of the response; possible families are *binomial, Gamma, Gaussian, inverse-gaussan* and *poisson* (optional: default is *gaussian*).
- *data* is the name of a data frame from which the variables referred to in the model statement are to be extracted (optional).
- *subset* allows fitting to a subset of the data (optional).
- *control* is a list of parameters controlling the iterative fitting procedure (optional). You only need to change these from defaults if something goes wrong!

Example:

$> hills.glm <- glm(time \sim distance + climb, family = gaussian, data = hills)$

fits the model

$$time = \beta_0 + \beta_1 distance + \beta_2 climb + \varepsilon$$

assuming that $time \sim N(\mu, \sigma^2)$. This is equivalent but less efficient than using *lm*.

Output is an object of type *glm*: a number of functions are available to act on objects of this type, e.g. *summary, plot, etc.* See the help system, for example via

$$>?glm$$

for more details.