GPCET
Pioneering Innovative Education

G. PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY

Accredited by NAAC with 'A' Grade of UGC, Approved by AICTE, New Delhi

Permanently Affiliated to JNTUA, Ananthapuramu

(Recognized by UGC under 2(f) and 12(B) & ISO 9001:2008 Certified Institution)

Nandikotkur Road, Venkayapalli, Kurnool – 518452

Department of Computer Science and Engineering

Bridge Course
On
R Programming

By

Mr. B. Venkateswarlu

GPCET
Pioneering Innovative Education

Table of Contents

Sno	Topic	Page number
1	Introduction and preliminaries	1
2	Simple manipulations; numbers and vectors	2
3	R Basics 1. Data Input 2. Missing Values 3. Useful Functions 4. Subsets of Dataframes 5. Scatterplots	3

1. Introduction and preliminaries

1.1 The R environment

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. Among other things it has

- ✓ an effective data handling and storage facility,
- ✓ a suite of operators for calculations on arrays, in particular matrices,
- ✓ a large, coherent, integrated collection of intermediate tools for data analysis,
- ✓ graphical facilities for data analysis and display either directly at the computer or on hardcopy, a well developed, simple and effective programming language (called 'S') which includes conditionals, loops, user defined recursive functions and input and output facilities. (Indeed most of the system supplied functions are themselves written in the S language.)

The term "environment" is intended to characterize it as a fully planned and coherent system, rather than an incremental accretion of very specific and inflexible tools, as is frequently the case with other data analysis software.

R is very much a vehicle for newly developing methods of interactive data analysis. It has developed rapidly, and has been extended by a large collection of packages. However, most programs written in R are essentially ephemeral, written for a single piece of data analysis.

1.2 Related software and documentation

R can be regarded as an implementation of the S language which was developed at Bell Laboratories by Rick Becker, John Chambers and Allan Wilks, and also forms the basis of the S-Plus systems.

The evolution of the S language is characterized by four books by John Chambers and coauthors. For R, the basic reference is The New S Language: A Programming Environment for Data Analysis and Graphics by Richard A. Becker, John M. Chambers and Allan R. Wilks. The new features of the 1991 release of S are covered in Statistical Models in S edited by John M. Chambers and Trevor J. Hastie. The formal methods and classes of the methods package are based on those described in Programming with Data by John M. Chambers.

There are now a number of books which describe how to use R for data analysis and statistics, and documentation for S/S-Plus can typically be used with R, keeping the differences between the S implementations in mind.

1.3 R and the window system

The most convenient way to use R is at a graphics workstation running a windowing system. This guide is aimed at users who have this facility. In particular we will occasionally refer to the use of R on an X window system although the vast bulk of what is said applies generally to any implementation of the R environment.

Most users will find it necessary to interact directly with the operating system on their computer from time to time. In this guide, we mainly discuss interaction with the operating system on UNIX machines. If you are running R under Windows or macOS you will need to make some small adjustments.

Setting up a workstation to take full advantage of the customizable features of R is a straightforward if somewhat tedious procedure, and will not be considered further here. Users in difficulty should seek local expert help.

1.4 Using R interactively

When you use the R program it issues a prompt when it expects input commands. The default prompt is '>', which on UNIX might be the same as the shell prompt, and so it may appear that nothing is happening. However, as we shall see, it is easy to change to a different R prompt if you wish. We will assume that the UNIX shell prompt is '$'.

In using R under UNIX the suggested procedure for the first occasion is as follows:

1. Create a separate sub-directory, say work, to hold data files on which you will use R for this problem. This will be the working directory whenever you use R for this particular problem.

```
$ mkdir work
$ cd work
```

2. Start the R program with the command

```
$ R
```

3. At this point R commands may be issued (see later).

4. To quit the R program the command is

```
> q()
```

At this point you will be asked whether you want to save the data from your R session. On some systems this will bring up a dialog box, and on others you will receive a text prompt to which you can respond yes, no or cancel (a single letter abbreviation will do) to save the data before quitting, quit without saving, or return to the R session. Data which is saved will be available in future R sessions.

Further R sessions are simple.

1. Make work the working directory and start the program as before:

```
$ cd work
$ R
```

2. Use the R program, terminating with the q() command at the end of the session.

To use R under Windows the procedure to follow is basically the same. Create a folder as the working directory, and set that in the Start In field in your R shortcut. Then launch R by double clicking on the icon.

2. Simple manipulations; numbers and vectors

2.1 Vectors and assignment

R operates on named data structures. The simplest such structure is the numeric vector, which is a single entity consisting of an ordered collection of numbers. To set up a vector named x, say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7, use the R command

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

This is an assignment statement using the function c() which in this context can take an arbitrary number of vector arguments and whose value is a vector got by concatenating its arguments end to end.

A number occurring by itself in an expression is taken as a vector of length one.

Notice that the assignment operator ('<-'), which consists of the two characters '<' ("less than") and '-' ("minus") occurring strictly side-by-side and it 'points' to the object receiving the value of the expression. In most contexts the '=' operator can be used as an alternative.

Assignment can also be made using the function assign(). An equivalent way of making the same assignment as above is with:

> assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))

The usual operator, <-, can be thought of as a syntactic short-cut to this.

Assignments can also be made in the other direction, using the obvious change in the assignment operator. So the same assignment could be made using

> c(10.4, 5.6, 3.1, 6.4, 21.7) -> x

If an expression is used as a complete command, the value is printed and lost 2. So now if we were to use the command

> 1/x

the reciprocals of the five values would be printed at the terminal (and the value of x, of course, unchanged).

The further assignment

> y <- c(x, 0, x)

would create a vector y with 11 entries consisting of two copies of x with a zero in the middle place.

2.2 Vector arithmetic

Vectors can be used in arithmetic expressions, in which case the operations are performed element by element. Vectors occurring in the same expression need not all be of the same length. If they are not, the value of the expression is a vector with the same length as the longest vector which occurs in the expression. Shorter vectors in the expression are recycled as often as need be (perhaps fractionally) until they match the length of the longest vector. In particular a constant is simply repeated. So with the above assignments the command

> v <- 2*x + y + 1

generates a new vector v of length 11 constructed by adding together, element by element, 2*x repeated 2.2 times, y repeated just once, and 1 repeated 11 times.

The elementary arithmetic operators are the usual +, -, *, / and ^ for raising to a power. In addition all of the common arithmetic functions are available. log, exp, sin, cos, tan, sqrt, and so on, all have their usual meaning. max and min select the largest and smallest elements of a vector respectively. range is a function whose value is a vector of length two, namely c(min(x), max(x)). length(x) is the number of elements in x, sum(x) gives the total of the elements in x, and prod(x) their product.

Two statistical functions are mean(x) which calculates the sample mean, which is the same as sum(x)/length(x), and var(x) which gives

sum((x-mean(x))^2)/(length(x)-1)

or sample variance. If the argument to var() is an n-by-p matrix the value is a p-by-p sample covariance matrix got by regarding the rows as independent p-variate sample vectors. sort(x) returns a vector of the same size as x with the elements arranged in increasing order; however there are other more flexible sorting facilities available (see order() or sort.list() which produce a permutation to do the sorting).

Note that max and min select the largest and smallest values in their arguments, even if they

are given several vectors. The parallel maximum and minimum functions pmax and pmin return a vector (of length equal to their longest argument) that contains in each element the largest (smallest) element in that position in any of the input vectors.

For most purposes the user will not be concerned if the "numbers" in a numeric vector are integers, reals or even complex. Internally calculations are done as double precision real numbers, or double precision complex numbers if the input data are complex.

To work with complex numbers, supply an explicit complex part. Thus

sqrt(-17)

will give NaN and a warning, but

sqrt(-17+0i)

will do the computations as complex numbers.

2.3 Generating regular sequences

R has a number of facilities for generating commonly used sequences of numbers. For example 1:30 is the vector c(1, 2, ..., 29, 30). The colon operator has high priority within an expression, so, for example 2*1:15 is the vector c(2, 4, ..., 28, 30). Put n <- 10 and compare the sequences 1:n-1 and 1:(n-1).

The construction 30:1 may be used to generate a sequence backwards.

The function seq() is a more general facility for generating sequences. It has five arguments, only some of which may be specified in any one call. The first two arguments, if given, specify the beginning and end of the sequence, and if these are the only two arguments given the result is the same as the colon operator. That is seq(2,10) is the same vector as 2:10.

Arguments to seq(), and to many other R functions, can also be given in named form, in which case the order in which they appear is irrelevant. The first two arguments may be named from=value and to=value; thus seq(1,30), seq(from=1, to=30) and seq(to=30, from=1) are all the same as 1:30. The next two arguments to seq() may be named by=value and length=value, which specify a step size and a length for the sequence respectively. If neither of these is given, the default by=1 is assumed.

For example

> seq(-5, 5, by=.2) -> s3

generates in s3 the vector c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0). Similarly

> s4 <- seq(length=51, from=-5, by=.2)

generates the same vector in s4.

The fifth argument may be named along=vector, which is normally used as the only argument to create the sequence 1, 2, ..., length(vector), or the empty sequence if the vector is empty (as it can be).

A related function is rep() which can be used for replicating an object in various complicated ways. The simplest form is

> s5 <- rep(x, times=5)

which will put five copies of x end-to-end in s5. Another useful version is

> s6 <- rep(x, each=5)

which repeats each element of x five times before moving on to the next.

2.4 Logical vectors

As well as numerical vectors, R allows manipulation of logical quantities. The elements of a logical vector can have the values TRUE, FALSE, and NA (for "not available", see below). The first two are often abbreviated as T and F, respectively. Note however that T and F are just variables which are set to TRUE and FALSE by default, but are not reserved words and hence can be overwritten by the user. Hence, you should always use TRUE and FALSE.

Logical vectors are generated by conditions. For example

> temp <- x > 13

sets temp as a vector of the same length as x with values FALSE corresponding to elements of x where the condition is not met and TRUE where it is.

The logical operators are <, <=, >, >=, == for exact equality and != for inequality. In addition if c1 and c2 are logical expressions, then c1 & c2 is their intersection ("and"), c1 | c2 is their union ("or"), and !c1 is the negation of c1.

Logical vectors may be used in ordinary arithmetic, in which case they are coerced into numeric vectors, FALSE becoming 0 and TRUE becoming 1. However there are situations where logical vectors and their coerced numeric counterparts are not equivalent, for example see the next subsection.

2.5 Missing values

In some cases the components of a vector may not be completely known. When an element or value is "not available" or a "missing value" in the statistical sense, a place within a vector may be reserved for it by assigning it the special value NA. In general any operation on an NA becomes an NA. The motivation for this rule is simply that if the specification of an operation is incomplete, the result cannot be known and hence is not available.

The function is.na(x) gives a logical vector of the same size as x with value TRUE if and only if the corresponding element in x is NA.

> z <- c(1:3,NA); ind <- is.na(z)

Notice that the logical expression x == NA is quite different from is.na(x) since NA is not really a value but a marker for a quantity that is not available. Thus x == NA is a vector of the same length as x all of whose values are NA as the logical expression itself is incomplete and hence undecidable.

Note that there is a second kind of "missing" values which are produced by numerical computation, the so-called Not a Number, NaN, values. Examples are

> 0/0

or

> Inf - Inf

which both give NaN since the result cannot be defined sensibly.

In summary, is.na(xx) is TRUE both for NA and NaN values. To differentiate these, is.nan(xx) is only TRUE for NaNs.

Missing values are sometimes printed as <NA> when character vectors are printed without quotes.

3. R Basics

3.1 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:

```
> library(DAAG)
> datafile(file="fuel") # NB datafile, not dataFile
```

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.

Exercise 2

The files molclock1.txt and molclock1.txt are in the data directory on the DVD.
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.

3.2 Missing Values

Exercise 3

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 4

For each column of the data frame Pima.tr2 (*MASS*), determine the number of missing values.
3 Useful Functions

Exercise 5

The function dim() returns the dimensions (a vector that has the number of rows, then number of columns) of data frames and matrices. Use this function to find the number of rows in the data rames tinting, possum and possumsites (all in the *DAAG* package).

Exercise 6

Use the functions mean() and range() to find the mean and range of:
(a) the numbers 1, 2, . . . , 21
(b) the sample of 50 random normal values, that can be generated from a normaL distribution with mean 0 and variance 1 using the assignment y <- rnorm(50).
(c) the columns height and weight in the data frame women.

[The *datasets* package that has this data frame is by default attached when R is started.]
Repeat (b) several times, on each occasion generating a nwe set of 50 random numbers.

Exercise 7
Repeat exercise 6, now applying the functions median() and sum().

3.4 Subsets of Dataframes
Exercise 8
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 9
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.

3.5 Scatterplots
Exercise 10
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 11**
Use of the argument log="xy" to the function plot() 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")
> ## Use lm() to fit a line, and abline() to add it to the plot
> Acmena10.lm <- lm(log10(wood) ~ log10(dbh), data=Acmena)
> abline(Acmena10.lm)
> ## Now print the coefficents, for a log10 scale
> coef(Acmena10.lm)
> ## For comparison, print the coefficients for a natural log scale
```

> 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 12

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) # orings has 23 rows

> 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 13

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.