

# An Introduction to R

Ian Dryden

## 1 Introduction

### 1.1 What is R?

R is a free version of a commercial statistical language/package called S-Plus. The commands in R and S-Plus are almost identical.

There is a huge range of statistical commands in R, from basic summary statistics to cutting edge research applications. It is also a programming environment, so you can soon build up your own routines.

Almost all practical statistical analysis is carried out on a computer, and so becoming familiar with computer packages and languages is an essential task for any statistician.

Please go through this tutorial at your own pace.

### 1.2 Starting R on University of South Carolina PC Network

First of all log into the MS network, and click on the R icon with the right mouse button twice.

R is a command line based language, so you will need to type in the commands with any options and arguments.

When R starts a command window is opened. It is here that you type in the commands.

Note that if you have your own PC then R can be downloaded from CRAN (Comprehensive R Archive Network)

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

You can also find additional documentation on R, including introductory material, at the CRAN site.

### 1.3 Starting R on a Linux/Unix network

In order to start R on the Linux or Unix network just type R.

## 2 Some basics

Let us start by typing in some data

```
x <- c(1,5,3,7,6,2,3,6,8,9)
```

Here the `c()` means it's a column vector, and the `<-` sign is an assignment (equivalent to `=`) The above command will put the vector of data  $(1, 5, 3, 7, 6, 2, 3, 6, 8, 9)^T$  into the variable named 'x'. You can see what is in 'x' by just typing

```
x
```

Now let us create a vector of length 60 consisting all of zeroes.

```
y <- rep(0,times=60)
```

And then we can look at it by just typing 'y'

```
y
```

Note that the numbers in square brackets indicate which element of the vector comes next.

Let's now consider simulating 10000 observations from a standard normal  $N(0, 1)$  distribution.

```
x <- rnorm(10000)
```

and we can look at some summary statistics for the data

```
summary(x)
mean(x)
sd(x)
var(x)
median(x)
```

We can view a graphical display

```
hist(x)
hist(x,nclass=100)
boxplot(x)
```

Adding comments: if we type

```
#
```

in a line then everything after that point is treated as a comment

```
# comments like this are ignored in R
```

If we want to simulate 10000 observations from a  $N(\mu = 2, \sigma^2 = 9)$  distribution then the command is

```
y <- rnorm(10000,2,3) # NB standard deviation is the third argument
```

We'll look at this command in more detail. Type

```
help(rnorm)
```

to obtain details of the command. You'll notice the help lists details of some other commands that involve the normal distribution (dnorm, pnorm, qnorm, as well as rnorm), and it gives the full details of the arguments for the command rnorm in the form `rnorm(n, mean=0, sd=1)`

When we issued the above command `rnorm(10000, 2, 3)` we were specifying that  $n = 10000$  (sample size),  $\text{mean}=2$ ,  $\text{sd}=3$ . We could have used the long-winded version of the command

```
rnorm(n=10000, mean=2, sd=3)
```

which is the same as `rnorm(10000, 2, 3)`

When we issued the first simulation command `rnorm(10000)` we didn't specify what the mean and sd were, so R uses the default values of 0 and 1, which can be found using `help(rnorm)`, or by typing the name of the command without brackets

```
rnorm
```

**The most important command of all is** `help(commandname)`

```
help.start()
```

is also useful for getting an overview of the package.

### Handy hint

If you press the up arrow then you get the last thing that you typed in. If you press it again you get the previous command to that. This is particularly useful if you had a small typo in a command or are carrying out lots of similar commands.

## 3 Simple graphics

### 3.1 Plots

We might want to plot our simulated values in  $y$  versus those in  $x$ .

The command is

```
plot(x, y)
```

[Note the  $x$  comes first before the  $y$ ]. There are numerous options for the plots that can be used, e.g. specifying the axes limits

```
plot(x, y, xlim=c(-4, 4), ylim=c(-15, 15))
```

Let's add the point (0,1) to the plot, which is coloured red ( $\text{col}=2$ ), and given a different plotting symbol ( $\text{pch}=2$ , is a triangle)

```
points(0, 1, col=2, pch=2)
```

All the different graphical parameters are found by looking at `help(par)`, and there are a lot of them!!

If you wish to remove variables from memory then use `rm`, e.g.

```
rm(x)
```

To list what variables you have in the memory then type

```
ls()
```

There is a huge list of commands in R, and in this session just a few can be explored.

## 3.2 Saving plots and text

You will usually need to write up your statistical analysis in a report, including some of the plots. One way for doing this is to open up a Word Document, then when in R press the right button when your mouse is over a plot, then select COPY as bitmap or metafile, then go to the Word Document and Paste it in.

Similarly you can highlight text in the commands window, copy, then paste that into the Word Document. Don't forget to save your Word document.

Alternatively, you can save the plots as Postscript files and then incorporate them in to a LaTeX document.

# 4 Reading in Data

## 4.1 Data

Most people find reading in the data in R a bit of a challenge at first. Datasets will often be posted on the web and then they can be saved on to your local disk space.

Consider the calcium data `calcium.txt` on the course website which contains three columns of data on English and Welsh towns: annual mortality rate per 100,000 males averaged over the years 1958-1964 in each town, the calcium concentration (in parts per million) in the drinking water supply for each town, and an indicator variable stating if the town in South (0) or at least as North (1) as Derby. There are many ways to read this dataset in, but perhaps the simplest is `read.table()`

```
z <- read.table("Z:/DirectoryName/calcium.txt")
```

This will put the data into a dataframe of the same size of the data (here 61 x 3)

Note the directory signs are forwards / rather than the usual backwards in Windows. So, in this case the file `calcium.txt` would be located on the PC in the directory

```
Z:\DirectoryName
```

, but this will of course be different for you.

To look at the data just type

```
z
```

and you will see it has rows labelled 1-61, columns labelled V1,V2,V3. A dataframe is just a matrix with some names labelling the columns. We can change the names if we want

```
names(z) <- c("mortality", "hardness", "north")
```

and then see what the dataframe looks like

```
z
```

We can now plot mortality against hardness using

```
plot(z$hardness, z$mortality)
```

Another good way to read data is from a comma separated .csv file.

```
z <- read.csv("Z:/DirectoryName/calcium.csv")
```

## 5 Statistical analysis

### 5.1 Linear models

There is a huge range of statistical functions that one can use in R.

We can fit a regression line using

```
out <- lm(z$mortality ~ z$hardness)
```

and the results, including the fitted line, tests for existence of regression etc. are put into the R object called 'out'. To get a summary of the results type

```
summary(out)
```

Is there evidence for existence of regression here? There are many other useful things you can do with the output, e.g. add the fitted regression line to the plot

```
abline(out)
```

Another way to read data is through the scan command

```
x <- scan("M:/G14CST/calcium.txt")
```

which scans in the data line by line and puts it all in a single column.

```
x
```

But now you'll need to re-arrange the data to put it into a matrix.

```
y <- matrix( x, 3, 61)
y <- t(y)
```

which puts the data into a 3 x 61 matrix first (you must do it this way round) and then we take the transpose of y to get a 61 x 3 matrix, which can be seen by typing

```
y
```

Let's plot the data again, but this time colour the northern and southern towns differently:

```
mortality<-y[,1]
calcium<-y[,2]
north<-y[,3]
plot(calcium,mortality,col=(north+2))
out<-lm(mortality~calcium+north+calcium:north)
summary(out)
```

This fitted linear model has mortality as the response, and calcium as a covariate with north as a factor and the interaction term is present, i.e. we have fitted two separate regression lines for the north and south data. Is there evidence for different regression lines?

## 5.2 Probability Plots

A useful plot is a Q-Q plot, where the ordered data ('Sample quantiles') plotted against the expected 'Theoretical' quantiles. The most common type of plot is a normal QQ plot.

Consider simulating 1000 observations from a Student's t distribution with 5 degrees of freedom.

```
x<-rt(1000,df=5)
```

We can then consider a normal qqplot of the data:

```
qqnorm(x)
```

or a Student's t Q-Q plot:

```
plot(sort(x),qt(ppoints(x),5))
```

## 6 Other things

### 6.1 Matrices

Let us look at some more matrix based commands

```
data(iris)
```

```
x <- iris
```

(assigns Fisher's Iris data into x, which is a standard dataset in R).

```
y <- x[, 1:4]
```

extracts the first four columns from the dataset. Note that we can add matrices A+B but if we multiply matrices we need to use the special symbol %\*\*%

Example:

```
z<-as.matrix(y) # converts dataframe to a matrix
A<-t(z)**z
dim(A)          # the dimension of A - here 4 x 4
```

where t(z) is the transpose of z.

To get the 4 x 4 sample covariance matrix we can use

```
S <- var(y)
S
```

and the correlation matrix is

```
R <- cor(y)
R
```

We can find the eigenvalues and eigenvectors of the symmetric matrix S as follows

```
eigen(S,symmetric=TRUE)
```

## 6.2 Lists

Note that the output of the previous command is a list, which has two parts

```
$values
```

(the eigenvalues)

and

```
$vectors (the eigenvectors)
```

We can create a list as follows:

```
ans <- list(x=0,y=0,z=" ")
```

where the list contains

```
$x (real), $y (real)
```

and

```
$z {
```

(character strings)

So, I can make an assignment

```
ans$x <- c(1,2,3,4,5)
ans$y <- c(4,3,2,1)
ans$z <- c("cat","dog")
```

and the list can be seen by typing

```
ans
```

Lists can be very useful in providing the output of functions.

## 7 Programming

### 7.1 Functions

A function is a series of commands that can be called by using a simple name and arguments. Consider the following function:

```
testfun<-function(n=100){
x<-rnorm(n)
xsumsq<-sum(x**2)
xsum<-sum(x)
name<- "Normal"
par(mfrow=c(1,2))
out<-list(sum=0,sumsq=0,name="")
hist(x)
hist(x**2)
out$sum <- xsum
out$sumsq <- xsumsq
out$name <- name
out
}
```

In order to see the details of the stored function, just type the function name

```
testfun
```

Now run the function for n=200 and put the answer in temp

```
temp<-testfun(200)
temp
```

Now run the function for the default value of n (100 here) and put the answer in temp2

```
temp2<-testfun()
temp2
```



## 7.2 Loops

Loops are very useful things for repeating calculations, although they can be very slow in R. The format is as follows for a simple loop for squaring the numbers 1-100 and printing out the values:

```
for (i in 1:100){
print(i**2)
}
```

## 7.3 If statements

You may need to check a logical statement with an IF statment. This loop does the same things but only for integers that are divisible by 3:

```
for (i in 1:100){
if (i/3==trunc(i/3)){
print(i**2)
}
}
```

## 7.4 While statements

Another useful thing is the WHILE loop, where a statement is carried out repeatedly while the condition is TRUE. If the condition is FALSE then the command is not carried out, and the program then exits the WHILE loop.

```
sum<-1
while (sum <= 100){
print(sum**2)
sum<-sum+1
}
print("finished!")
```

# 8 Finally...

## 8.1 Reading in R commands from a file

You can read in commands from a file using the 'source' command. For example, to read in from a file called 'example.txt':

```
source("Z:/DirectoryName/example.txt")
```

## 8.2 Finishing and saving your work

To quit a session of R then type

```
q( )
```

You'll be asked if you want to save the 'workspace image'. In general I would say 'no' to this question, unless you have been doing some very heavy numerical calculations that will take a long time to repeat. If you say 'yes' then next time you start R the variables will all be restored. This is fine for small jobs but when you start doing lots of different jobs it can be confusing to have unnecessary variables lying around.

If you wish to save your work then I would recommend writing the commands in a text file as you go along (e.g. using wordpad or notepad) and then copying and pasting them into the R window (or using the 'source' command as in the previous section.)

## 8.3 Installing packages

One of the many great things about R is that there is a huge number of additional packages for specialist applications. These packages have been contributed by members of the statistical community.

In Windows you can install a package by using the menu option 'Install package'.

In Linux/Unix you can install a package locally, or if you wish to use it a lot then it's worth requesting that it is added as a permanently available resource.

After the package has been installed you need to issue the command

```
library(packagename)
```

in order to use the commands. To find out about the commands type

```
library(help=packagename)
```

## 8.4 Some books of potential interest

Crawley, Michael J.(2005). Statistics : an introduction using R. Wiley.

Faraway, Julian J. (2005). Linear models with R. Springer.

Verzani, John (2005). Using R for introductory statistics. Chapman and Hall.

Venables, W. and Ripley, B.D. (2002). Modern Applied Statistics with S, 4th edition. Springer.