Introduction to R

Prepared by HongFei Li
Based on R manual http://www.r-project.org/

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, and
- 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.)

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 R and statistics

We think of R of an environment within which many classical and modern statistical techniques have been implemented. A few of these are built into the base R environment, but many are supplied as packages. There are about 25 packages supplied with R (called "standard" and "recommended" packages) and many more are available through the CRAN family of Internet sites (via http://CRAN.R-project.org) and elsewhere.

To install packages, use command (Windows)
installpackages('packagename')

1.3 Getting help with functions and features

R has an inbuilt help facility. To get more information on any specific named function, for example solve, the command is
help(solve)

An alternative is
?solve

On most R installations help is available in HTML format by running

help.start()
which will launch a Web browser that allows the help pages to be browsed with hyperlinks.

1.4 R commands, case sensitivity, etc.

Technically R is an expression language with a very simple syntax. It is case, so A and a are different symbols and would refer to different variables. The set of symbols which can be used in R names depends on the operating system and country within which R is being run (technically on the locale in use). Normally all alphanumeric symbols are allowed (and in some countries this includes accented letters) plus '.' and '_', with the restriction that a name must start with '.' or a letter, and if it starts with '.' the second character must not be a digit.

Elementary commands consist of either expressions or assignments. If an expression is given as a command, it is evaluated, printed, and the value is lost. An assignment also evaluates an expression and passes the value to a variable but the result is not automatically printed.

Commands are separated either by a semi-colon (';'), or by a newline. Elementary commands can be grouped together into one compound expression by braces ('{' and '}'). Comments can be put almost anywhere, starting with a hashmark ('#'), everything to the end of the line is a comment.

If a command is not complete at the end of a line, R will give a different prompt, by default
+
on second and subsequent lines and continue to read input until the command is syntactically complete. This prompt may be changed by the user. We will generally omit the continuation prompt and indicate continuation by simple indenting.

2. R Basics

2.1 Elementary commands

Elementary commands consist of either expressions or assignments. For example,
42 + 8

Instead of just evaluating an expression, we can assign the value to a scalar using
x <- 42 + 8
x

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 a alternative.

2.2 Vectors

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, 5, 3, 6 and 21, use the R command

```
x <- c(10, 5, 3, 6, 21)
x
```

The further assignment

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

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

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
v
```

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(x)  # the largest elements of a vector x

min(x)  # the smallest elements of a vector x.

range(x)  # a vector of length two, namely c(min(x), max(x)).

length(x)  # the number of elements in x,

sum(x)  # the total of the elements in x.

mean(x) # sample mean.

var(x)  #sample variance.

sd(x) # standard deviation
```

Character quantities and character vectors are used frequently in R, for example as plot labels. Character strings are entered using either double (") or single (’) quotes.

```
lev <- c("high","low")
lev
```

2.3 Generating regular sequences

```
1:6   # c(1, 2, 3, 4, 5, 6)

x <- seq(1,6)    # seq(from=1, to=6)
x
seq(-5, 5, by=.2)

s1 <- rep(x, times=5)  # put five copies of x end-to-end in s1.
s1
s2 <- rep(x, each=5)  # repeats each element of x five times before moving on to the next.
s2
```

Exercise: 1. set up a vector named u, consisting of 3, 4, 5, 9, 2, 7, 5, 3, 2.
2. set up a vector named v, consisting of 1, 1, 1, 1, 4, 5, 6, 7, 8.
3. find the mean of u.
4. find the log value of each of the eight numbers of u.
5. find the range of u.
6. calculate u+2*v

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). The first two are often abbreviated as T and F, respectively.

```
temp <- x > 3
temp
```

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.

```
temp1 <- x>3 & x<5
temp1
```

2.5 Missing values

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(0:3,NA)
z
ind <- is.na(z)
ind
```

2.6 Index vectors; selecting and modifying subsets of a data set

Subsets of the elements of a vector may be selected by appending to the name of the vector an index vector in square brackets.

```
y <- z[!is.na(z)]
y
```

Values corresponding to TRUE in the index vector are selected and those corresponding to FALSE are omitted.

```
u <- z[(!is.na(z)) & z>0]
u
```

creates an object u and places in it the values of the vector z for which the corresponding value in z was both non-missing and positive.

```
z[1:3]   # selects the first 3 elements of z (assuming length(x) is not less than 3).
z[c(1,3)]    # selects the first and the third elements of z

z[-1]  # remove the first element

fruit <- c(5, 10, 1, 20)
names(fruit) <- c("orange", "banana", "apple", "peach")
 lunch <- fruit[c("apple","orange")]
```

The advantage is that alphanumeric names are often easier to remember than numeric indices. This option is particularly useful in connection with data frames, as we shall see later.

```
z[is.na(z)] <- 0  # replaces any missing values in z by zeros
```

Exercise: 1. set up a vector named u, consisting of 5, 6, -1, NA, 4.
2. find the mean of all positive numbers.

2.7 Matrices

Matrix objects are frequently needed and can be created by the use of the matrix() function. The general syntax is

matrix(data, nrow, ncol, byrow=F)

```
mat.1 <- matrix(seq(1:10),nrow=2)
mat.1
mat.2 <- matrix(seq(1:10),nrow=2,byrow=T)
mat.2

dim(mat.1)          ## the dimension
mat.1[1,1]   # The first number specifies the row, and the second specifies the column.
mat.1[1,]         ## first row
mat.1[,1:2]           ## the first two columns

mat.3 <- mat.1[-1,]   ## remove the first line
mat.3

mat.1%*%t(mat.2)   #matrix multiplication
mat.1*mat.2           #element by element multiplication
```

2.8 Data frame

Data frames can bind vectors of different types together (for example, numeric and character), retaining the correct type of each vector. In other respects, a data frame is like a matrix so that each vector should have the same number of elements.

```
price <- c(5, 10, 1, 20)
fruit <- c("orange", "banana", "apple", "peach")
 lunch <- data.frame(fruit, price)     # create a data frame
lunch
lunch$price    # the second column of the data frame
lunch[,2]      # the second column of the data frame
lunch[,'price']     # the second column of the data frame
```

Exercise:

The following is a hypothetical data set giving sex, ages, weights and heights of five individuals.

Individual	Sex	Age	Weight(lbs)	Height(inches)
1	M	20	120	61
2	F	24	130	65
3	M	31	190	70
4	M	25	201	74
5	F	27	101	71

1. Read the data into a data frame, Data.
2. Find the mean weight of the women in Data.

2.9 Read data from files

Large data objects will usually be read as values from external files rather than entered during an R session at the keyboard. If variables are to be held mainly in data frames, as we strongly suggest they should be, an entire data frame can be read directly with the read.table() function.

```
tox <- read.table("toxicity.dat ")
tox <- read.table("toxicity.dat ", header=TRUE)
```

where the header=TRUE option specifies that the first line is a line of headings, and hence, by implication from the form of the file, that no explicit row labels are given.

2.10 Plot function

```
names(tox)
tox$con
names(tox) <- c('d','n','r')
tox[1:2,]
tox$p <- tox$r/tox$n * 100     ## percentage killed
tox

plot(tox$d,tox$p)              # plot d versus p
plot(tox$d, tox$p,main="Toxicity Data",ylab="% killed",xlab="Concentration")
lines(tox$d, tox$p)
# add title for the figure and x axis, y axis

par(mfrow=c(1,2))
plot(tox$d, tox$p,type="b")
plot(tox$d, tox$p,type="b",lty=3,lwd=3,pch=16,cex=2)
# change the line type, line width, point size.

hist(tox$d)          # histogram
boxplot(tox$d)       # boxplot
```

Exercise:

Based on the exercise in section 2.8, use the data frame, Data, and do the following questions:

1. Produce a diagram containing the boxplots of Age, Weight and Height, and scatterplots of Age vs Weight, Age vs Height and Weight vs Height.
2. Use the appropriate dialogue box to test whether the mean weight of men is different from that of women.

3. Fit a linear regression model

3.1 Fit a linear regression model

```
lakes.data <- read.table("lakes.dat",header=T)

dim(lakes.data)
summary(lakes.data)

#Change the variable names
names(lakes.data)
names(lakes.data) <- c("Case","CH","PH","NT")
names(lakes.data)

#Fit a linear regression model of CH on PH and NT
fit1 <- lm(CH ~ PH+NT, data=lakes.data)
summary(fit1)

#Getting summary objects
names(fit1)
fit1$coefficients
```

3.2 Assessing the fit of the model

```
#Extract residual object
resids1 <- fit1$residuals
resids1.std <- resids1/fit1.summary$sigma        #standarized residuals

#assess normality of the residuals
qqnorm(resids1.std)
qqline(resids1.std)

#scatterplots of the residuals
par(mfrow=c(3,1))
plot(fit1$fitted.values,resids1.std,ylab="Std. Residuals",xlab="Fitted Values")
abline(h=0)
plot(lakes.data$PH,resids1.std,ylab="Std. Residuals",xlab="Phosphorus")
abline(h=0)
plot(lakes.data$NT,resids1.std,ylab="Std. Residuals",xlab="Nitrogen")
abline(h=0)
```

Exercise:

1. Fit and evaluate the linear model log(CH) regressed on log(PH) and NT
2. Fit and evaluate the linear model log(CH) regressed on log(PH), NT and NT^2