Getting Started with R

Marc Schleiss and Alexis Berne
Laboratoire de Télédétection Environnementale,
École Polytechnique Fédérale de Lausanne, Switzerland

1 INTRODUCTION

R is a high-level language and environment for data analysis and visualization. Its applications range from elementary data handling and exploratory statistics up to more advanced topics such as spatial statistics, multivariate statistics and generalized mixed models. Today, a large proportion of the world's leading statisticians use R and contribute to its development. Last but not least, the R product is completely free and multiplatform.

To run R, simply click on the R icon on your desktop or in the Programs menu. The first thing you see is the version number and date of R. Before starting, make sure your version is up-to-date. Below the header you will see a blank line starting with a ">" symbol. This is called the prompt and is basically where you type your commands. When working, you will sometimes see a "+" instead of ">", meaning that the last command you entered is incomplete. If you press the "Esc" key or "Ctrl+C", the command line ">" will reappear.

The simplest way to get help in R is to click on the Help button on the toolbar of the RGui window. If you work at the command line prompt, simply type "?" followed by the name of the function you want help with, e.g., "?read.table" to see how to read data. Alternatively, if you do not know the exact name of the function, you can use the help.search() command followed by the query in double quote, e.g., "help.search("data input")". Finally, there is tremendous amount of information about R on the web.

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

If you want to install additional libraries, you can do so by using the install.packages() command followed by the name of the package in double quotes, e.g., "install.packages("gstat")".

2 Command line versus scripts

When writing functions and multi-line commands, you will find it useful to use a text editor rather than executing everything directly at the command line. You can use R's own built-in editor or any other text editor of your choice. The script editor is accessible from the RGui menu bar. Click on "File" then on New Script to open a new script. To execute a line or a group of lines, just highlight them and press "Ctrl+R". Alternatively, you can select the lines and copy-paste them into the terminal. Popular alternatives to R's built-in editor are: "Rstudio" (http://www.rstudio.com/ide) and "Tinn-R"(http://www.sciviews.org/Tinn-R).

3 ESSENTIALS

3.1 Value assignment

Objects obtain values using the "<-" operator. Thus, to create a scalar constant x with value 1, type "x <- 1". Do not use the "=" sign to avoid confusion with the comparison operator "==".

3.2 Display the value of a variable

If you want to display the value of a variable, simply type the name of this variable in the command line and press "ENTER". When running a script, you can use the print() function, e.g., "print(x)" to display the value of x.

3.3 Arithmetical operators

Basic arithmetical operators are $+, -, *, /, \hat{}$. Most common mathematical functions like "log" (logarithm in base e), "exp" (exponential), "sqrt" (square root) as well as trigonometric functions (sin, cos, tan) are also implemented. The function abs() returns the absolute value. Different integer rounding functions like round(), floor() and ceiling() can also be used. Finally, integer quotients and remainders are obtained by using the "%/%" and "%%" operators.

3.4 Vectors

Vectors are variables with one or more values of the same type. In particular, scalar variables are vectors of length 1. Note that empty vectors have length zero. You can either create a vector by explicitly giving its elements separated by commas, e.g., "v <- c(1,2,3)", or you can create them using the sequence generator operator "v <- 1:3". To concatenate two vectors v and w, simply type "c(v,w)". Particular elements of a vector can be accessed using square brackets. For example, "v[2]" returns the 2nd element of v. Note that several elements can be accessed at the same time. For example, "v[1:2]"

returns the 1st and the 2nd elements of v while "v[c(1,3)]" returns the 1st and the 3rd elements. Finally, the size of a vector can be computed by typing "length(v)".

3.5 Matrices

Similarly to vectors, matrices can be created through enumeration of their individual elements:
> M <- matrix(c(1,0,0,0,2,0,0,0,3),nrow=3)
creates a 3× 3 matrix given by

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

where by default the elements are entered columnwise. By typing "dim(M)" you will get the dimensions of M, i.e., the number of rows and columns:
> dim(M)
[1] 3 3
Individual elements of a matrix can be accessed by specifying their position using subscripts. Since matrices have two dimensions, two subscripts are necessary (one for the row and one for the column).
> M[1,2]
[1] 0
> M[2,2]
[2] 2
> M[c(1,3),1]
[3] 1 0
You can also access a full column or a full row by omitting the corresponding subscript.
> M[1,]
[1] 1 0 0
> M[,2]
[2] 0 2 0
Sometimes, it is useful to give names to the rows and the columns of a matrix. This can be done using rownames() and colnames().
> colnames(M) <- c("height","weight","age")
You can use the names to extract a given row or a given column.
> M[,"height"]
[1] 1 0 0
Finally, two matrices M and N can be multiplied by typing "$M\%*\%N$". Be carefull because this is not the same than "$M*N$" which performs element-by-element multiplication.

3.6 Logicals

Logical expressions evaluate either in TRUE (1) or FALSE (0). Relational operators are given by $<, >, , ==, <=, >=, !=$. Logical operators are given by !,&,| (not,and,or). They can be applied to single elements or to vectors:
> 3==4
[1] FALSE
> c(3,4)==4
[2] FALSE TRUE

3.7 Vector functions

One of the great strengths of R is its ability to evaluate functions over entire vectors, thereby avoiding the need for loops and subscripts. Important vector functions are sum, cumsum, prod, diff, match, min, max, range, mean, median, var, sd, cor, quantile, sort, which. For matrices you can use colSums, rowSums, colMeans, rowMeans and many other...

For example, to count the number of elements of a vector v that are smaller than 5, simply type "sum($v < 5$)". To extract all elements of v that are smaller than 5, type "v[which($v < 5$)]".

3.8 Character strings

In R, character strings are defined by double quotation marks, i.e., "s <- "hello"". The function nchar() returns the number of characters forming a string. The function substr() can be used to extract substrings of a specified length. For the purpose of printing, you might want to suppress the quotes that appear around strings by default. This can be done using the noquote() function. If you want to concatenate two strings s1 and s2, simply write "paste(s1,s2,sep="")" where "sep" gives the character used to separate the two strings. Note that by default, the separator is a single blank space.

3.9 Data frames

Reading and handling data is one of the most important issues when dealing with R. By default, R handles most of its data in objects called data frames. A data frame is an object with rows and columns (like a matrix). But unlike matrices which can only contain numbers, data frames can contain any type of objects like character strings, logicals, dates and times and of course numbers. The key thing about working effectively with data frames is to become completely at ease with using subscripts. In R, subscripts appear in square brackets []. Since a data frame is a two dimensional object with rows and columns, the rows are referred to by the first subscript and the colums by the second. Hence "data[3,5]" returns the value on row 3 and column 5. To select all the entries in the 3rd row, type "data[3,]".

3.10 Input/Output

To read data from an external text or excel file using R, simply type "read.table(file_name)". If the file is not located in the current working directory, you will have to specify its absolute path. Make sure the file name is enclosed in double quotes, e.g., "c:\\abc.txt" with the correct extension and always use double backslash \\ rather than \ in the file path definition. When writing directly in the command line, you can use "read.table(file.choose())" which allows you to select the desired file by opening a browser window. Hint: If you work frequently in the same path you can define it as your current working directory using the setwd() function, e.g., "setwd("c:\\abc")".

It is often useful to generate numbers with R and to save them somewhere else. To do this, you can use the write() function. Be careful if you want to save a table or a matrix because write() transposes the rows and columns.

4 PLOTS

Generating plots with R is very easy and straightforward. The plot() command allows to create traditional two variables scatterplots. Some extra functions named text(), points(), lines(), abline(), rect() can be used to add text, points, lines or rectangles to an existing plot. The following examples should illustrate this procedure.

Example 1: two variable plots

For this example, we use a standard data set called "hills" which is included in the very popular "MASS" package.

> library(MASS)
> attach(hills)

The data set consists of record times in 1984 for 35 Scottish hill races. There are three main variables for each race: "dist" (distance in miles), "climb" (total height gained during the race in feet) and "time" (record time in minutes).

To plot the time with respect to the distance type:

> plot(dist,time)

Note that by default the axes of the plot are labeled with the variable names, unless you choose to override these with "xlab" and "ylab". To add a line with intercept -4.8 and slope 8.3, type

> abline(-4.8,8.3,col="red")

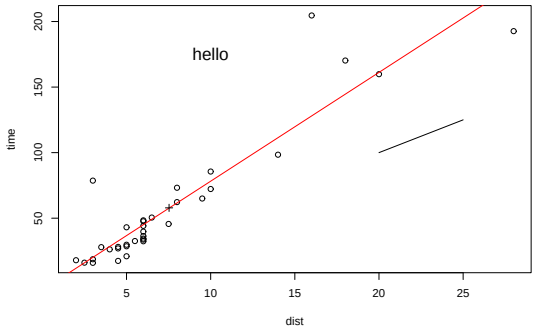
Additional points can be added using the points() function. To change the type of points (dots, crosses, circles, ...), you can use the "pch" option. Several points can be added by specifying a vector of coordinates. The "l" option allows to draw lines between the points.

> points(mean(dist),mean(time),pch=3)
> points(c(20,25),c(100,125),"l")

To add some text to your graph, simply use the text() function. The "cex" option allows to change the size of the text.

> text(10,175,"hello",cex=1.5)

If you typed all the commands above, here is what you should obtain:



Of course, there are many other useful graphical parameters and functions available. Please read the "help(par)" and "help(plot)" to get a list of graphical options available for your plots.

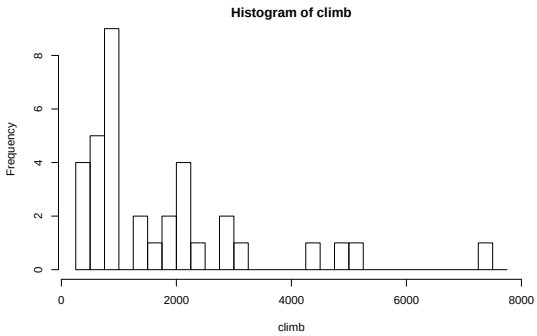
Example 2: histograms

Histograms are the most common plots used to show the frequency distribution of a given variable. They are excellent for showing the mode, the spread and the symmetry of a data set. To display a histogram, you can use the hist() function.

> library(MASS)
> attach(hills)
> hist(climb)

You should always specify the break points on the x axis yourself. Unless you do this, the hist function may not take your advice about the number of bars or the width of the bars.

> hist(climb,breaks=seq(250,7750,250))



By default, hist() shows the individual counts (in

each class) of your variables. If you rather want to plot the empirical density (total area equal to 1), you can do that by using the "freq" option.

```
> hist(climb,freq=FALSE)
```

Example 3: plot into a file

By default, all plots are displayed in what is called a "graphical device". All devices are associated with a name (e.g. "X11" or "postscript") and numbered from 1 to 63. The function dev.new() can be used to open a new graphical device. Once a device has been opened, it is placed into a circular list. The functions dev.next() and dev.prev() can be used to select the next or the previous device. The function dev.off() shuts off an open device. By default, R opens and closes the devices automatically. But sometimes it happens that a device stays open (e.g. after an incorrect call of plot) and you will have to shut it down manually. This is particularly important when plotting into a file instead of the screen. The default device for generating plots is "X11" (the screen). Unless specified, all the plots will appear on the screen and will not be saved. If you want to create a png, an eps or a pdf file out of your plot, you have to specify it by writing "png(file_name)", "postscript(file_name)" or "pdf(file_name)" where the name of your file must be given in double quotes with the correct extension, e.g., pdf("C:\\abc\\picture.pdf"). Next time the plot() function is called, nothing appears on the screen. Instead, a file is generated in the specified directory. Additional points or lines can be added to the plot until the current device is shut down.

```
> pdf("C:\\abc\\picture.pdf")
> plot(x,y)
> points(p,q)
> ...
> dev.off()
```

5 THE GSTAT PACKAGE

There are numerous packages and libraries for spatial statistics and analysis in R but gstat (http://www.gstat.org) is THE reference for calculation, fitting and visualization of direct and cross-variograms. It can deal with a large number of practical issues like change of support (block kriging), simple/ordinary/universal (co)kriging, fast local neighborhood selection and efficient simulation of large correlated random fields. Most commonly used functions in gstat are called variogram, fit.variogram, vgm, krige and predict.gstat. Please read the help for more information on how to use these functions.

6 ADVANCED FEATURES

Like many other high-level languages, R allows you to write your own functions and routines. The syntax for writing a function is

function <- name(argument list){body}

The argument list is a comma-separated list of formal arguments, i.e., a symbol, a variable, a function or a statement of the form symbol=expression. The body can be any valid R expression or set of R expressions. At the end of the body, the statement return() is used to return a value or an object. The following examples should help:

Example 1:

```
function < - is.odd(n){
    if(n%%2==0){return(FALSE)}
    else{return(TRUE)}
}
```

Example 2:

```
function < - inverse_string(s){
    n <- nchar(s)
    inv <- substr(s,n,n)
    for(i in 1:(n-1))
        inv <- paste(inv,substr(n-i,n-i),sep="")
    }
    return(inv)
}
```

Once a function has been created, it can be used like any other built-in function, simply by calling its name and giving its arguments.

```
> is.odd(7)
[1] TRUE
> is.odd(6)
[2] FALSE
> inverse_string("hello")
[3] "olleh"
```

Finally, R provides a number of flow control statements which are very useful when programming your own functions. The most commonly used are "if... else" statements. To create loops, you can use "for", "while" or "repeat" statements. To break out of a loop, use the "break" statement. To continue with the next iteration, use "next". To stop the script, use "stop()". Interested readers are invited to read the help for more information on these flow control statements.