R tutorial, session 1

H. Seitz (IGH)
(herve.seitz@igh.cnrs.fr)

October 2, 2014

Contents

1 The R prompt

When you start an R session, a terminal will open, where you can type commands. The line where commands are to be typed starts with a ">" symbol. This character is called the "prompt" (when R displays it, it means it is expecting you to type a command):

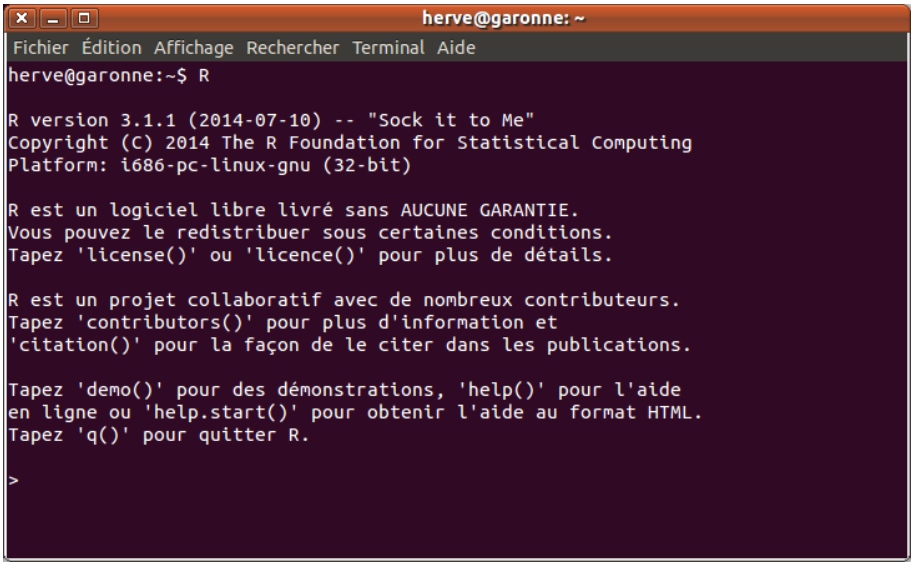


Figure 1: **The R prompt.**

You can then type a command, *e.g.*, a simple calculation like "2+2" and press Enter:

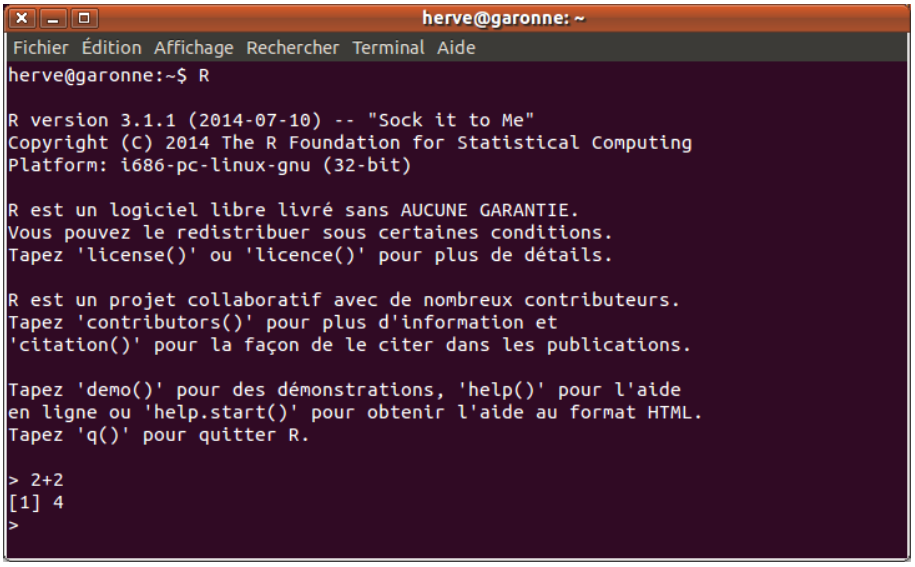


Figure 2: **Entering a command.** The result of the command is displayed in a line that does not start with the ">" character.

The result of the command ("4") is displayed on a line that does not start with the ">" prompt. Instead, that line starts with "[1]": we will see why in section 2.

R contains many mathematical functions, as well as some built-in constants (like the number π). Figure 3 shows some examples of functions readily usable in R.

Figure 3: **Mathematical function usage.** Here I calculated e^3, the natural logarithm of 28, the square root of 9, 2 to the power 5, $(1+\frac{3}{2})\times 4 - cosine(2)$, and I displayed the (approximate) value of π.

You can also define variables: they will be represented by letters (or words[1]). You may use either the "=" or the "<-" symbols to attribute a value to a variable. Once the variable is defined, type its name and press Enter: its value will be displayed:

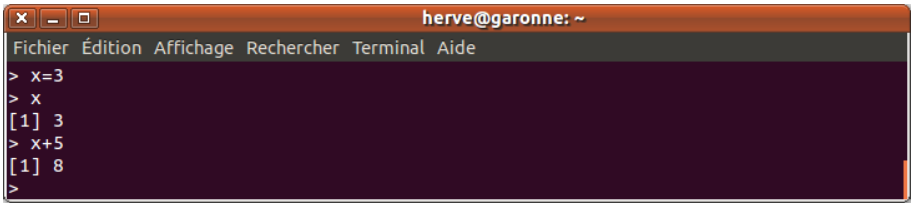


Figure 4: **Working with variables.** I defined a variable (named "x") and gave it the value: 3. Then I displayed its value, and I used that variable in a simple calculation: $x+5$.

Once you are done and you want to exit R, you type:
`q()`
("q" is for "quit", and the parentheses show that R considers "q" as a function; it is a very special function, with no argument, so the parentheses are empty, unlike "cos(2)" in figure 3, where the argument, 2, was given to the function "cos"). Then you press Enter, you will be asked whether R should save the values you have been working on (if you type "y", these values will be automatically restored at your next R session), then the R terminal closes.

[1]There are a few constraints on the names you can give to variables: they may not start with a digit, and they may not contain mathematical operators (like "+", "-", *etc.*): such variable names would be ambiguous to R.

2 Types of variables

You can define and manipulate several types of objects in R: figure 4 showed how to define and use a scalar variable (a "scalar" is a number). You can also define "vectors" of numbers: these are ordered lists of values:

Figure 5: **Vectors.** The first line defines a vector (named "a") using the command "c()" (elements in the vector are separated by commas). I then displayed the content of "a", then I displayed the second element of "a".

You can refer to a given element of a vector using square brackets: in figure 5, "a[2]" means: "the second element of vector a".

It is important not to confuse the two types of brackets: parentheses "(" and ")" are used to pass an argument to a function; square brackets "[" and "]" are used to extract an element in a vector.

If the vector contains many elements, displaying its content will span on several lines in the terminal:

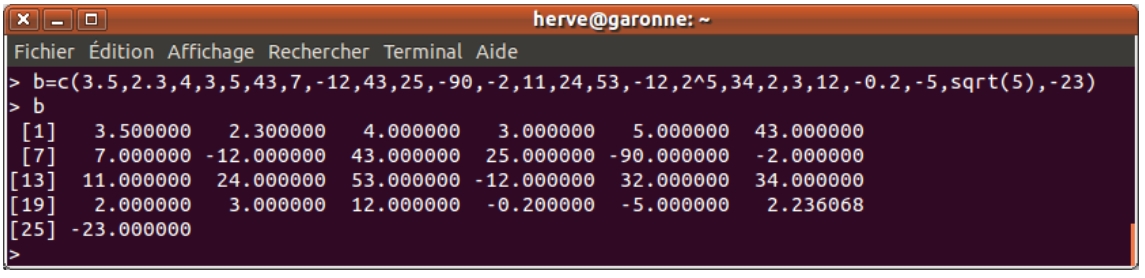


Figure 6: **Displaying long vectors.** When the vector is very long, its elements span on several lines. Each line starts with the rank of the first displayed elements, between square brackets.

Each line here starts with a number between square brackets: that's the rank of the first displayed element in the vector (*e.g.*, 3.5 is the first element in the vector, 7 is the 7th element, 11 is the 13th element, *etc.*). This is the reason why the line displaying the value of variable x (see figure 4) started with "[1]": that's because a scalar variable is in fact a vector with a single element... and when it is displayed, R shows the rank of that variable in its single-element vector, which is "1".

You can also use multi-dimensional vectors (in other words: tables). They can be defined with command "array":

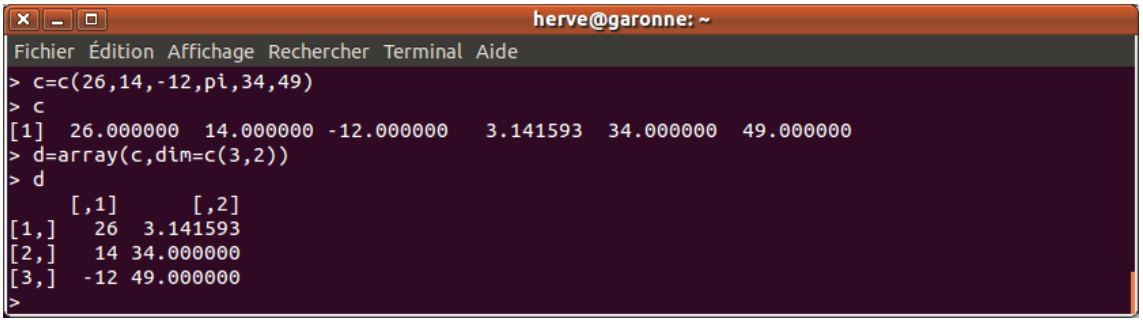


Figure 7: **Defining a bi-dimensional vector.** The command "array" converts a uni-dimensional vector into a multi-dimensional one.

Then you can refer to a specific element of a bi-dimensional array using square brackets and commas:

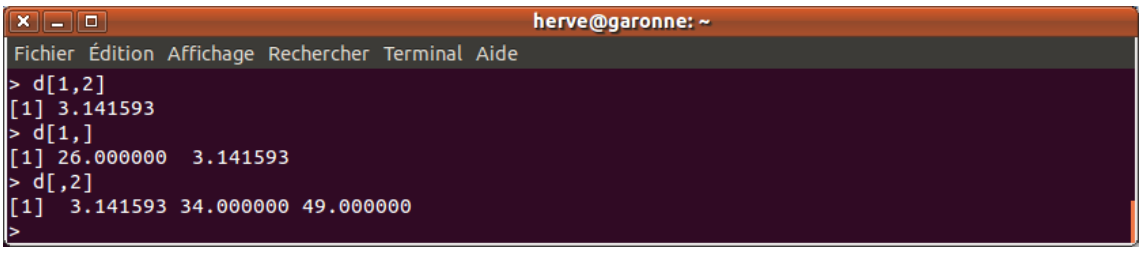


Figure 8: **Extracting elements, or vectors, from a multi-dimensional vector.** The value "d[1,2]" is the element on the first row, second column of table "d"; vector "d[1,]" is the first row, and vector "d[,2]" is the second column, of d.

It is also possible to define vectors of higher dimension (tridimensional, quadridimensional, *etc.*), though parsing their display on a terminal gets a bit tricky.

Of note: vectors do not have to be numbers. It is possible to define a vector of character strings for example:

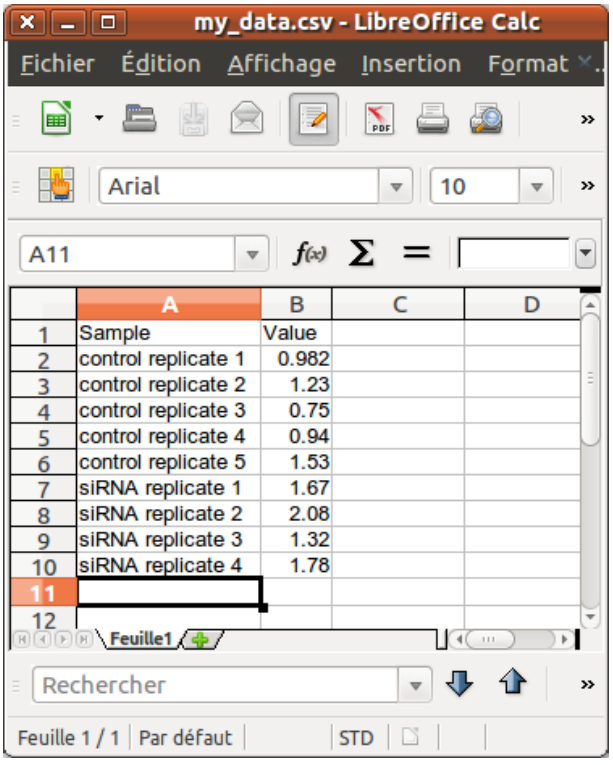
Figure 9: **A vector of character strings.**

Many other types of variables can exist in R; in particular, "lists" and "data.frames". These are tables whose columns do not all have to have the same nature (there can be columns of numbers, columns of character strings, *etc.*; this is impossible with arrays: all the columns in an array have to have the same type).

3 Loading data from an external file

Instead of entering values by hand in R, you can also load data from a text file. The input file may be in the ".csv" format (*N.B.*: Excel can save files in the CSV format), or in a tabulation-delimited, or space-delimited format (you can also ask Excel to save file as tabulation- or space-delimited files under the ".txt" format). It is also possible to load ".xls" files, but for that you need to install a specific R package (see section 5 for a description of package installation).

Let's assume you have a ".csv" file containing data that you want to import in R, and that file is name 'my_data.csv':



	A	B	C	D
1	Sample	Value		
2	control replicate 1	0.982		
3	control replicate 2	1.23		
4	control replicate 3	0.75		
5	control replicate 4	0.94		
6	control replicate 5	1.53		
7	siRNA replicate 1	1.67		
8	siRNA replicate 2	2.08		
9	siRNA replicate 3	1.32		
10	siRNA replicate 4	1.78		
11				
12				

Figure 10: **An example data file (to be saved in the ".csv" format).**

As long as the file is in the current directory (*i.e.*: the folder where R has been launched), it will be loadable using the "read.csv" function:

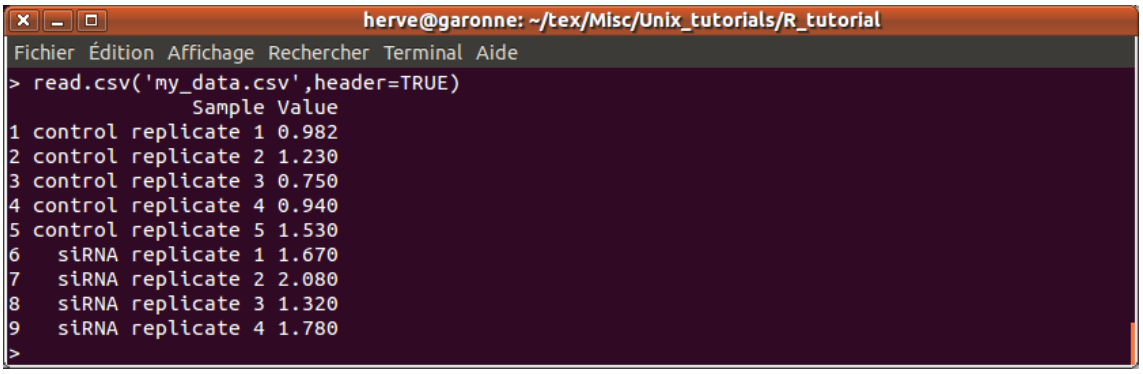


Figure 11: **Loading data from an external ".csv" file in R.**

The function "read.csv" takes one mandatory argument: the name of the ".csv" file. You can add optional arguments ("header" specifies whether the first line contains column names, or whether these are already data).

Just like for every other function in R, you can always get some help by typing:

```
?read.csv()
```

(it will display a manual for the usage of that function, as well as some details on available options, and examples of usage). You can quit that help page by typing:

```
q
```

(like "quit").

Instead of simply displaying the content of the file in the terminal, you can also save it in a variable. By default, that variable will be a "data.frame" (see section 2): in this example, it is an object that contains a column of character string (column "Sample") and a column of scalars (column "Value"). You can extract columns from a data.frame using the "$" sign, followed by the column title:

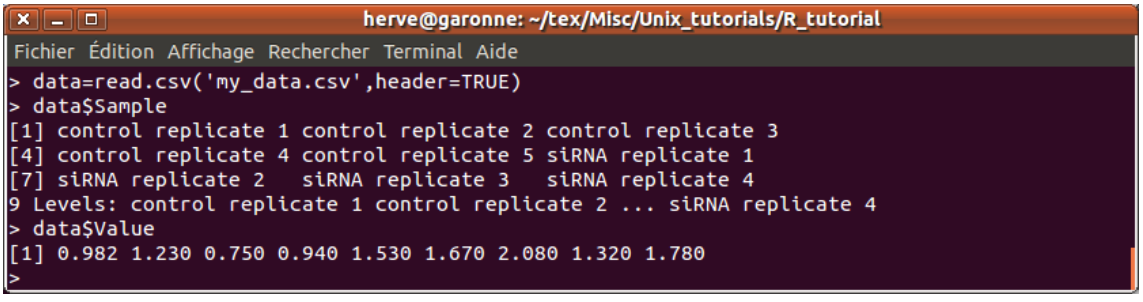


Figure 12: **Extracting data from a data.frame.**

Of note: each of the extracted column is a vector. In this example, "data$Sample" is a vector of character strings. When R displays vectors of character strings, it lists the values spanned by the vector (see the line starting by "9 Levels:" in figure 12).

4 Conditional selection

You can apply filters on vectors, either based on the index of the element in a vector, or based on their mathematical properties:

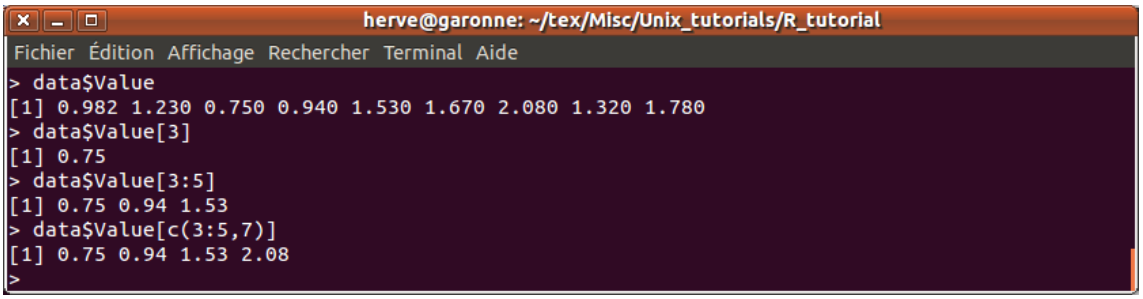


Figure 13: **Selecting vector elements using their index.**

We saw (in section 2) that `a[2]` is the second element of a vector named `a`. R uses the ":" symbol to indicate a range of values ("3:5" means: all the integer values between 3 and 5, in an increasing order; "5:3" means: all the integer values between 5 and 3, in a decreasing order). So `data$Value[3:5]` means: "The 3rd, 4th and 5th element of vector `data$Value`".

The last line of this example uses a more complex pattern: `c(3:5,7)` is a vector containing all the integer values between 3 and 5, then value 7 (so that vector is: 3, 4, 5, 7).

You may also select values in a vector by their mathematical properties:

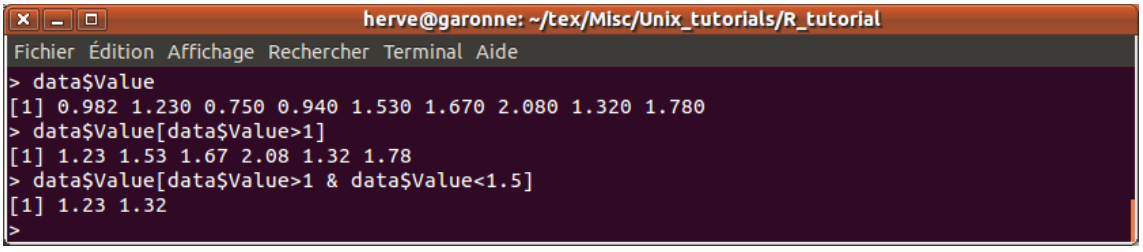


Figure 14: **Selecting vector elements using their mathematical properties.**

Filters are also written between square brackets (*e.g.*, `data$Value>1` will select all the values strictly larger than 1), and they can be combined (`data$Value>1 & data$Value<1.5` selects values between 1 and 1.5). Here, the "&" character means "AND", and the "—" character means "OR". You can add parenteses to group conditions:

`(a>1 & a<1.5) | (a<0 & a>-1.5)`

(that filter would select values for which a falls between 1 and 1.5 or between 0 and -1.5).

You can also select values from one vector, based on a filter applied to another vector:

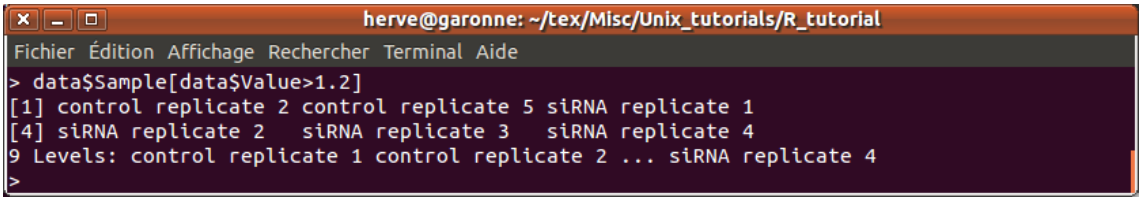


Figure 15: **Selecting vector elements using information from another vector**

Here, we selected the sample names (from vector `data$Sample`) based on a filter on the values in vector `data$Value`.

Such filtering scheme will make it very easy to select biological data for further processing (*e.g.*, selecting the fold-change of genes whose p-value is lower than a cutoff; selecting the sequence of small RNAs whose abundance exceeds a cutoff; *etc.*).

5 Adding more functions to R with additional packages

Some functions are not included in the basic R installation (for example, Levene's test, which we will use to test for the homogeneity of variances): they are provided by additional packages, that can be installed (a given package just needs to be installed once per computer: you won't have to re-install it every time you will use a function from that package).

For this, your computer has to be connected to the Internet. Most R packages are available from a central repository called "CRAN" (the comprehensive R archive network; `http://cran.r-project.org/`), which has mirror sites on every continent.

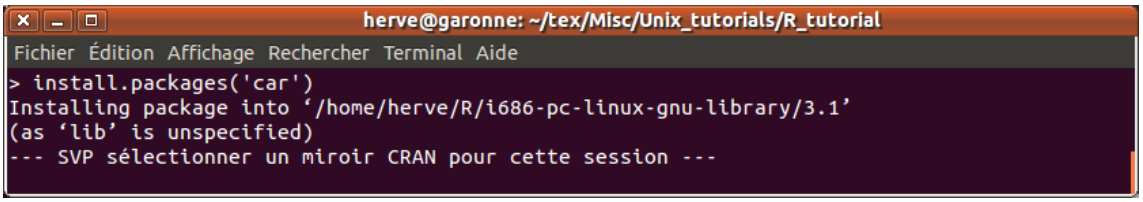


Figure 16: **Installing a package.** Here examplified with package 'car'.

Note that the package name has to be given between quotes (so it is not interpreted as a variable name). A list of CRAN mirror sites will show up (one of them is at the IGH, in Montpellier: it is always better to choose the closest server, geographically, to speed up data transfer). Once you chose a mirror site and clicked "OK", the download will start (it should take a few seconds).