Initiation to R software Session I

Pierre Michel

Master AMSE 1st year, 2019

Introduction

General information on R

- R (1995, AT&T Bell Laboratories) is a software for **statistical analysis** and **graphics**, it is a clone of S-PLUS, mainly written in C language.
- R is **free software**, distributed freely under the terms of GNU Public Licence of the Free Software Foundation (FSF). The development and distribution are ensured by several statisticians (R Development Core Team). It is **compatible with all platforms**.
- Files and instructions for installing R, tutorials and updates are available from the CRAN (Comprehensive R Archive Network).

Free softwares

The FSF has a definition of free software based on four **freedoms**:

1. The freedom to **run** the program, for all uses.
2. The freedom to **study** the functioning of the program (this requires access to the source code).
3. The freedom to **redistribute** copies (this includes the freedom to sell copies).
4. The freedom to **improve** the program and publish its improvements (this encourages the creation of a community of developers improving the software).

General information. . .

R programming language is:

1. interpreted: the **available functions** are located in a library (i.e a directory), organized in **packages** containing functions, operators and datasets.
2. object-oriented: variables, data, results in a R session are stored in the form of **objects** in memory in the **workspace**.

Try the following example:

```r
a = 1 # store variable a containing the value 1
a = a + 1 # add 1 to a
print(a) # print the value of a
```

Manage the working directory

To print the path to the **current** working directory (set by default) use the following command:

```
getwd()
```

the working directory can be assigned by the user using the following command:

```
setwd("C:/Users/myName") # Windows example
```

These two functions are very useful to manage your different projects in R.

Tips and tricks to start using R

The first thing to do is to run R (type `R` in a terminal on Unix, or run the Rgui on Windows). You have now access to the R console. Below are some tips and tricks for using R:

- the command `q()` quits R
- press Esc to interrupt R
- press ↑ and ↓ to recall last commands
- press ← and → to move the cursor on the command line
- multiple commands on the same line must be separated by `;`
- reserved words: `NA`, `letters`, `LETTERS`, `T`, `F`, `TRUE`, `FALSE`
- `n` is not the same object as `N`: R is case-sensitive
- run a R script using the command `source("myFile.R")`
- print a result: `print()`, use code comments: `#`

Manage the workspace

- Most of the commands you will use in R will **create objects**.
- To create an object, we use an **assignment operator**: <- or =.

```
# Try this example
n = 5
M = matrix(1:10, 2, 5)
N <- c(1, 2, 3)
h = n + N
n = 5 ; M = matrix(1:10, 2, 5); N <- c(1, 2, 3);
h = n + N
```

- To print the value of an object, type its name.
- To print all the objects created in the workspace, use: `objects()` or `ls()`.
- To remove objects, use: `remove()` or `rm()`.

Functions and operators

Functions and operators

- Objects that interact with other objects, stored in packages.
- Among the pre-installed packages, the package `base` proposes basis functions and operators for reading and handling data and some standard statistical and graphical functions.
- To list the loaded packages, use: `search()`.
- To list the available packages, use: `library()`.
- To load a package, use: `library("packageName")`.
- To list the functions of a package, use: `help(package = "packageName")`.

```r
# Try this example
library("survival")
help(package = "survival")
```

Functions and operators: `help()`

- To get help on R in html format, use: `help.start()`.
- To get help on a function from a loaded package, use: `help()` or `?`.

```
# Try this example
?"+"
help("*")
help(log)
help("log")
?sum
```

Note: Actually, this is one of the **most important function** for a R programmer, so I advise to **always** see the help on a function before using it.

Other objects

- The main object types are: `vector`, `factor`, `matrix`, `array`, `time-series`, `list` and `data.frame`
- An object is characterized by its **name**, its **content**, and two attributes:
  - `mode()` which can be numeric, character, logical or complex,
  - `length()` which gives the number of elements in the object.
- A missing value is represented by `NA` for any mode.

```
# Try this example
u = 1:3
v = letters[1:3]
w = c("a", "b", NA)
x = c(T,F)
mode(u); mode(v); mode(w); mode(x)
```

Vectors

Vectors

A vector is an ordered sequence of elements of the **same mode**.

Here is an example with a **numeric** vector:

```r
u = 1:5
print(u); mode(u); length(u)
```

```
## [1] 1 2 3 4 5
```

```
## [1] "numeric"
```

```
## [1] 5
```

Vectors

A vector is an ordered sequence of elements of the **same mode**.

Here is an example with a **character** vector:

```
v = c("a","b","c")
print(v); mode(v); length(v)
```

```
## [1] "a" "b" "c"
```

```
## [1] "character"
```

```
## [1] 3
```

Vectors

A vector is an ordered sequence of elements of the **same mode**.

Here is an example with a **logical** vector:

```r
w = c(T,F)
print(w); mode(w); length(w)
```

```
## [1]  TRUE FALSE
```

```
## [1] "logical"
```

```
## [1] 2
```

Creating vectors

To create a vector enter some values separated by , with the function `c()` (“c” stands for “concatenate”).

```r
u = c(1, 2, 3)
v = c("a","b","c")
w = c(T,F)
print(u); print(v); print(w);
```

```
## [1] 1 2 3

## [1] "a" "b" "c"

## [1]  TRUE FALSE
```

Creating numeric vectors

- `:` operator to create an ordered sequence.
- `seq()`: create a regular sequence.
- `rep()`: duplicate a sequence.
- `sample()`: create random vectors (see further).

```r
# Try this example
u = 1:10
v = seq(from = 0, to = 10, length = 11)
w = seq(from = 0, to = 2, by = 0.5)
t = rep(1:4, 2)
r = rep(1:4, each = 2)
print(u); print(v); print(w); print(t); print(r);
```

Creating character vectors

- `letters`: letters in lower cases (from a to z)
- `LETTERS`: letters in upper cases (from A to Z)

```
u = letters
v = LETTERS
u[3]
```

```
## [1] "c"
```

```
v[c(1,2,4)]
```

```
## [1] "A" "B" "D"
```

Creating random vectors

The function `sample()` draws random samples, with or without replacement, uniformly or according to a discrete distribution.

```r
u = sample(1:10)
v = sample(1:10, 3)
w = sample(1:2, 5, replace = T)
x = sample(1:4, 3, prob = c(0.1, 0.2, 0.6, 0.1))
print(u); print(v); print(w); print(x);
```

```
##  [1]  9  6  4  1  3  5  8  2  7 10
```

```
## [1]  3 10  7
```

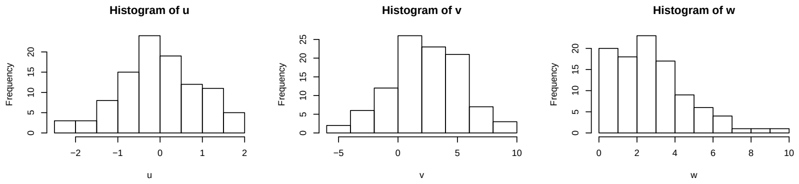
```
## [1] 2 1 1 2 2
```

```
## [1] 2 3 1
```

Creating random samples

The user can draw random samples from a given distribution, in the following example we only consider normal and Poisson distributions, so do not hesitate to see the other available distrubtions.

```r
u = rnorm(100) # normal distribution N(0,1)
v = rnorm(100, 2, 3) # normal distribution N(2,3)
w = rpois(100, 3) # Poisson distribution P(3)
par(mfrow = c(1,3)); hist(u); hist(v); hist(w);
```



Element-wise vector operations

- arithmetic: +,-,*,/,^
- comparison: >, <, >=, <=, ==, !=
- logical: &, |, !

```
# Try this example
v1 = 1:2
v2 = c(T, F)
v3 = letters[c(1, 2)]
v4 = rep(1:2, 2)
v5 = v1 + v2
v6 = v1 + v4
v7 = v1 / v3
v8 = v1 < v2
v9 = v1 ^ 2
v10 = v1 == v2
v11 =! v10
```

Global vector operations

`identical()` and `all.equal()` functions

```
v1 = 1:2; v2 = c(T, F)
w = identical(v1, v2) # for all mode vectors
x = all.equal(v1, v2) # for numeric vectors
v2 = as.numeric(v2) # converts to numeric
print(w); print(x);
```

```
## [1] FALSE
```

```
## [1] "Modes: numeric, logical"
## [2] "target is numeric, current is logical"
```

```
all.equal(v1, v2)
```

```
## [1] "Mean relative difference: 1"
```

Global vector operations: `paste()`

This function aims to change vectors in character vectors, and "paste" them element by element. The function `paste0()` does the same thing, without a space between elements.

```r
x1 = 0:2
x2 = c("a", "b", "c")
paste(x1, x2)
```

```
## [1] "0 a" "1 b" "2 c"
```

```r
paste0(x1, x2)
```

```
## [1] "0a" "1b" "2c"
```

Selecting vector elements by position indices

Position **indices** can be used for vectors using `[]`.

```
v = -3:2
v[3]
```

```
## [1] -1
```

```
a = c(4,6)
v[a]
```

```
## [1] 0 2
```

Selecting vector elements with boolean vectors

A boolean vector can contain only the value `TRUE` or `FALSE`, it can be used to select elements in a vector of the **same size**.

```
v = -3:2
l=c(F, F, F,F,T,T); v[l]
```

```
## [1] 1 2
```

```
l = v > 0; v[l]
```

```
## [1] 1 2
```

Selecting vector elements with boolean vectors: `which()`

The `which()` function returns the positions for those the logical indicator is `TRUE`.

```
v = -3:2
l = which(v > 0); v[l]
```

```
## [1] 1 2
```

Selecting vector elements with negative indices

Negative position indices can be used to **deselect** corresponding elements.

```
v = -3:2
w = v[-4]
w
```

```
## [1] -3 -2 -1  1  2
```

Additional information on vectors

If two vectors have not the same length, arithmetic and comparison operations return a vector of the same length as the longest one, and duplicating the shortest one.

```r
v = 1:2; w = 1:4; v + w;
```

```
## [1] 2 4 4 6
```

To see a vector mode, use the functions `is.numeric()`, `is.logical()`, or `is.character()`.

```r
v = 1:2; is.logical(v); is.numeric(v)
```

```
## [1] FALSE
```

```
## [1] TRUE
```

Additional information on vectors

To change a vector mode, use the functions `as.numeric()`, `as.logical()`, or `as.character()`.

```
u = c("1", "d", "T"); as.numeric(u)
```

```
## [1]  1 NA NA
```

```
v = 0:2; as.logical(v)
```

```
## [1] FALSE  TRUE  TRUE
```

```
w = 1:3; as.character(w)
```

```
## [1] "1" "2" "3"
```

Factors

Factors

A factor is a **categorical variable** containing two attributes:

- a vector of **values**
- a vector of **levels**

It must contain elements of same mode.

```r
factor(1:4)
```

```
## [1] 1 2 3 4
## Levels: 1 2 3 4
```

```r
factor(letters[c(1,3,4)], levels=c("a","b","c","d"))
```

```
## [1] a c d
## Levels: a b c d
```

Creating factors

```r
factor(letters[c(1,3,4)], levels=c("a","b","c"))
```

```
## [1] a    c    <NA>
## Levels: a b c
```

To create a factor, use the function `factor()`. The functions `is.factor()` and `as.factor()` are also available.

```r
factor(1:3); factor(1:3, levels=1:5)
```

```
## [1] 1 2 3
## Levels: 1 2 3
```

```
## [1] 1 2 3
## Levels: 1 2 3 4 5
```

Creating factors

```r
factor(1:3, exclude=2);
```

```
## [1] 1    <NA> 3
## Levels: 1 3
```

```r
factor(1:3, labels=c("a", "b", "c"))
```

```
## [1] a b c
## Levels: a b c
```

```r
x=1:3; is.factor(x); as.factor(x)
```

```
## [1] FALSE
```

```
## [1] 1 2 3
## Levels: 1 2 3
```

Creating factors: `cut()`

Divides the range of a numeric vector in *n* classes et encodes the values according to the class they belong. The resulting object is a factor.

```r
v=1:10; cut(v, breaks=2)
```

```
##  [1] (0.991,5.5] (0.991,5.5] (0.991,5.5] (0.991,5.5] (0.991,5.5]
##  [6] (5.5,10]    (5.5,10]    (5.5,10]    (5.5,10]    (5.5,10]
## Levels: (0.991,5.5] (5.5,10]
```

```r
cut(v, breaks=c(1,5.5,10))
```

```
##  [1] <NA>     (1,5.5]  (1,5.5]  (1,5.5]  (1,5.5]  (5.5,10] (5.5,10]
##  [8] (5.5,10] (5.5,10] (5.5,10]
## Levels: (1,5.5] (5.5,10]
```