Introductory course on the R software

P. Lafaye de Micheaux[1]

[1]Mathematics and Statistics Departement
Université de Montréal
CANADA

New South Wales University
Sydney, Australia
May 13, 2014

https://biostatisticien.eu/springeR/courseRw2.pdf

Goals of today lecture

Introducing the

- basics concepts of the **R** software :
  - calculator mode
  - assignment operator
  - variables
  - functions
  - arguments
- various data types and structures which can be handled by **R**.

R is a calculator

R can easily replace all the functionalities of a (very sophisticated !) calculator.

```
> sin(2*pi/3)      # <--- this symbol is for comments.
[1] 0.8660254
> 5^2              # Same as 5**2.
[1] 25
> sqrt(4)          # Square root of 4.
[1] 2
> log(1)           # Natural logarithm of 1.
[1] 0
> c(1,2,3,4,5)     # Collection of the first 5 integers.
[1] 1 2 3 4 5
> c(1,2,3,4,5)*2   # First five even numbers.
[1]  2  4  6  8 10
```

Displaying results and variable redirecting

R responds to your requests by displaying the result obtained after evaluation. **But, this result is displayed, then lost.**
Nevertheless, we can use the **assignment arrows** : <- or ->

```
> x <- 1          # Assignment.
> x               # Display.
[1] 1
> 2 -> x          # Assignment (in the other direction).
> x               # Display.
[1] 2
> (x <- 1)        # Assignment AND display.
[1] 1
```

The = symbol

It is not good practice to use the = symbol for assignment.

Continuation symbol

If a command is not complete at the end of a line, R will display a different prompt symbol, by default the plus sign (+), on the second line and on following lines. R will continue to wait for instructions until the command is syntactically complete.

```
> 2*8*10+exp(1)
[1] 162.7183
> 2*8*
+ 10+exp(
+ 1)
[1] 162.7183
```

Rules for choosing a variable name

- a variable name can only include alphanumerical characters as well as the dot (.) ;
- variable names are *case sensitive*, which means that **R** distinguishes upper and lower case ;
- a variable name may not include white space or start with a digit, unless it is enclosed in quotation marks "".

Work strategy

You should use either a *script window* (R *editor*) or Rstudio to type your commands before sending them to R for execution.

The key combinations `CTRL+R` (or `CTRL+ENTER`) can be used to execute your commands.

You can also use the `source(file.choose())` command (typed in the R console) to read and execute the contents of an external R script file.

Note : The function `help()` can be used to access the documentation : `help(source)`.

Perform the two "Do it yourself" on pages 42 and 43.
`http://biostatisticien.eu/springeR/Rbook-chap3.pdf`

Using functions

A function in R is defined by its **name** and by the list of its **parameters** (or **arguments**). Most functions output a **value**.

Using a function (or **calling** or **executing** it) is done by typing its name followed, in brackets, by the list of (formal) arguments to be used. Arguments are separated by commas. Each argument can be followed by the sign = and the value to be given to the argument.

```
functionname(arg1=value1,arg2=value2,arg3=value3)
```

Note that you do not necessarily need to indicate the names of the arguments, but only the values, as long as you follow their order. For any R function, some arguments must be specified and others are optional (because a default value is already given in the code of the function).

Understanding the use of arguments

The function `log(x,base=exp(1))` can take two arguments : `x` (its value must be specified) and `base` (optional, because a default value is provided as `exp(1)`).

You can call a function by playing with the arguments in several different ways. This is an important feature of R which makes it easier to use.

```
log(3)
log(x=3)
log(x=3,base=exp(1))
log(x=3,exp(1))
```

```
log(3,base=exp(1))
log(3,exp(1))
log(base=exp(1),3)
log(base=exp(1),x=3)
```

Question : what is done with this instruction ?

```
log(exp(1),3)
```

Using functions

Don't forget the brackets when you call a function

A common mistake for beginners is forgetting the brackets :

```
> factorial # Typing the name gives the code.
function (x)
gamma(x + 1)
<environment: namespace:base>
> factorial(6)
[1] 720
> date
function ()
.Internal(date())
<environment: namespace:base>
> date() # Brackets are also necessary when
>        #  no arguments are required.
[1] "Wed Jan  9 16:04:32 2013"
```

Creating functions

It is very easy to code a new function in R, by using the function `function()`.

For example, here is how to code a function which takes two arguments n and p and calculates the binomial coefficient

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

```
> binomial <- function(n,p) factorial(n)/(factorial(p)*
+                                 factorial(n-p))
```

You can now then use this new function as any other R function :

```
> binomial(4,3)
[1] 4
```

The various data types in R

One of the main strengths of R is its ability to organize data in a structured way. **This will turn out to be very useful for many statistical procedures.**

Data type	Type in R	Display
real number (integer or not)	`numeric`	`3.27`
complex number	`complex`	`3+2i`
logical (true/false)	`logical`	`TRUE` or `FALSE`
missing	`logical`	`NA`
text (string)	`character`	`"text"`
binary	`raw`	`1c`

Dealing with data type

```
> a <- 1 # Similar to: a <- 1.0
> typeof(a)
[1] "double"
> c <- as.integer(a)
> typeof(c)
[1] "integer"
> b <- 3.4
> c(b>a,a==b)
[1] TRUE FALSE
> is.numeric(a)
[1] TRUE
> is.integer(a)
[1] FALSE
> x <- TRUE # Similar to: x <- T
> is.logical(x)
[1] TRUE
```

Missing data

A missing or undefined value is indicated by the instruction `NA` (for *non available*).

```
> x <- c(3,NA,6)
> is.na(x)
[1] FALSE  TRUE FALSE
> mean(x)                    # Let's try to calculate the mean.
[1] NA
> mean(x,na.rm=TRUE)         # The na.rm argument indicates
                             # that NA's should be removed.
[1] 4.5
```

Character string type

Any information between quotation marks (single ' or double ") corresponds to a character string.

```
> a <- "R is my friend"
> mode(a)
[1] "character"
> is.character(a)
[1] TRUE
```

The various data structures in R

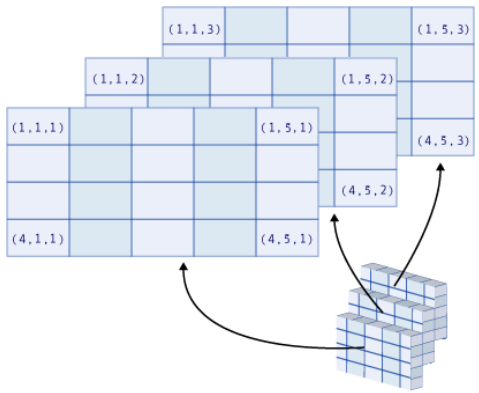
Data structure	Instruction in R	Description
vector	`c()`	Sequence of elements of the **same nature**.
matrix	`matrix()`	Two-dimensional table of elements of the **same nature**.
multidimensional table	`array()`	More general than a matrix ; table with several dimensions.
list	`list()`	Sequence of R structures of any (and possibly different) nature.
individual×variable table	`data.frame()`	Two-dimensional table. The columns can be of different natures, but must have the same length.
factor	`factor()`, `ordered()`	Vector of character strings associated with a modality table.
dates	`as.Date()`	Vector of dates.
time series	`ts()`	Values of a variable observed at several time points.

Vectors

```
> c(3,1,7)
[1] 3 1 7
> c(3,TRUE,7) # Automatic conversion occurs.
[1] 3 1 7
> seq(from=0,to=1,by=0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(from=0,to=20,length=5)
[1]  0  5 10 15 20
> vec <- 1:10 # Stored as integers.
> names(vec) <- letters[1:10]
 a  b  c  d  e  f  g  h  i  j
 1  2  3  4  5  6  7  8  9 10
> (vec <- 2:33) # [17] = rank of the next element.
 [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
[17] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
```

Matrices and arrays

```
> ( X <- matrix(1:12,nrow=4,ncol=3,byrow=TRUE) )
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> class(X)
[1] "matrix"
> X <- array(1:12,dim=c(2,2,3))
```



Lists : the most flexible and richest structure in R

Unlike the previous structures, lists can **group together in one structure data of different types** without altering them.

```
> ( A <- list(TRUE,-1:3,my.matrix=matrix(1:4,nrow=2),
+             c(1+2i,3),"A character string") )
[[1]]
[1] TRUE
[[2]]
[1] -1  0  1  2  3
$my.matrix
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[[4]]
[1] 1+2i 3+0i
[[5]]
[1] "A character string"
```

Note: we have named this element of A.

Data frames : most used structure in Statistics

```
> ( BMI <- data.frame(Gender=c("M","F","M","F","M","F"),
+     Height=c(1.83,1.76,1.82,1.60,1.90,1.66),
+     Weight=c(67,58,66,48,75,55),row.names=c("Jack",
+     "Julia","Henry","Emma","William","Elsa")) )
        Gender Height Weight
Jack         M   1.83     67
Julia        F   1.76     58
Henry        M   1.82     66
Emma         F   1.60     48
William      M   1.90     75
Elsa         F   1.66     55
> str(BMI) # Structure of each column.
'data.frame': 6 obs. of  3 variables:
 $ Gender: Factor w/ 2 levels "F","M": 2 1 2 1 2 1
 $ Height: num  1.83 1.76 1.82 1.6 1.9 1.66
 $ Weight: num  67 58 66 48 75 55
```

Factors and ordinal variables

```
> x <- factor(c("blue","green","blue","red",
+                         "blue","green","green"))
> x
[1] blue  green blue  red   blue  green green
Levels: blue green red
> levels(x)
[1] "blue"  "green" "red"
> class(x)
[1] "factor"
>z<-ordered(c("Small","Tall","Average","Tall","Average",
+  "Small","Small"),levels=c("Small","Average","Tall"))
> class(z)
[1] "ordered" "factor"
```

Your turn to work !

You can now try to do the **Exercises** and the **Worksheet** of Chapter 3.

`http://biostatisticien.eu/springeR/Rbook-chap3.pdf`