Stat4Astro2017

Introduction to R

Octobre 9, 2017

Didier Fraix-Burnet

1 Introduction

R is a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. It comprises libraries of nearly all the statistical literature. Please consult the R project homepage for further information (`http://www.r-project.org/`).

R is an object-oriented interpretated language, made of an ensemble of packages. Packages are installed through a compilation of codes C++ and FORTRAN (gcc compiler). As an interpreted language, it is very slow for *loops*.

R is used by most statisticians in the world mainly for research and development. There is even a package to manage FITS format: FITSio.

R is command-line driven, making it a wonderful tool for complex scripts and repeated procedures.

CRAN (Comprehensive R Archive Network) is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load.

There are many and many help websites and blogs (e.g. `http://cran.r-project.org/manuals.html` or `http://www.statmethods.net/`). Simply asking your question in your favorite internet search engine, you will surely find the corresponding script. To make a search on the web regarding R, type "r-cran" instead of "R"!

See `http://cran.r-project.org/doc/manuals/R-intro.html` for a complete introduction to R.

You can fin a Short Reference "Card" at https://cran.r-project.org/doc/contrib/Short-refcard.pdf that is also provided for this session.

A useful table of command equivalence for MATLAB users: `http://mathesaurus.sourceforge.net/octave-r.html`.

1.1 Installation of R

You can download a binary version for your platform: http://cran.r-project.org/ . For Linux users, you may check whether R is in the distribution.

Warning: once R is installed, if your are behind a proxy, it should be indicated with a line like:
`http_proxy='http://www-cache.ujf-grenoble.fr:3128/'`
in a file *~/.Renviron* or at the end of the file */etc/R/Renviron* to be able to install and update packages.

1.2 Interfaces GUI

A non-necessary but appreciable complement is to have a GUI interface to R which provides an integrated environment with the command line, the plot window and an editor for the script files. This is extremely useful to save commands into a file and execute any selection in this file

There is one included in the Mac binary, otherwise I would recommend RStudio (http://www.rstudio.org/). See all GIUs on http://sciviews.org/_rgui/.

1.3 Entering R in command mode

To open R, type *R*.

When in R , the prompt > will appear. It will not be indicated in the following command lines (to ease copy and paste operations...).

To quit:

```
q()
```

To obtain help:

```
help(plot)
?plot
??plot    # to find a command in a package which is not loaded yet
```

Help pages all have the same structures with description, usage, format (options), Details, Source and Examples at least. Do not hesitate to run the examples.

1.4 Using R in batch mode

Scripts can be launched in batch mode with the following command :

Linux:

```
 R CMD BATCH file.R file.Rout
```

Windows

```
"C:\Program Files\R\R-2.13.1\bin\R.exe" CMD BATCH --vanilla --slave "c:\my projects\file.R"
```

where file.R is the R script file and file.Rout is an optional file containing the outputs.

1.5 Packages

Packages are ensembles of programs, commands and scripts. They must be installed, i.e. compiled. Then they can be loaded to be used or detached when you are done.

Warning: commands of the newly installed packages hide commands with same name already in memory until the new package is uninstalled ("detached"). These commands are indicated when the package is loaded. Some commands automatically adapt their behaviour to the type of objects (for instance, plot can have several slightly different behaviours). For this reason, and except for some basic set of packages, loaded packages are not kept into memory after session's end.

Packages can be installed directly from a repository (you will be proposed a choice, see full list on http://cran.r-project.org/mirrors.html) or from a previously loaded file.

(To be executed).

To install packages:

```
install.packages("coda")
```

To load packages into memory and access its commands:

```
library(coda)
```

To remove a package from memory:

```
detach(package:coda,unload=T)
```

Sometimes it is useful to update packages:

```
update.packages(repos="https://mirror.ibcp.fr/pub/CRAN",ask=F,checkBuilt=T)
```

1.6 Files

R uses two important files which are saved under the current directory:

- *.Rhistory* with the history of the commands

- *.RData* the environment file made of the functions and objects in memory. The **loaded packages are not kept when you quit R** to avoid command confusions in the next session.

It is a good idea to have different directories for different projects. To change directory within R:

```
setwd("the_directory_path")
getwd()
```

2 Basics

2.1 Commands

Commands are always followed by closed parentheses. Otherwise it prints the content of the object.

```
ls()   # prints the result of the command
ls     # prints the command which in this case is a function
```

If parentheses are not closed, then the symbol + appears asking for a missing closing), } or ".

Two commands written on the same line have to be separated by “;”.

The function rm() removes (definitvely) an object from memory:

```
a <- "Hello" ; a ; rm(a) ; a
```

2.2 Objects

Objects have values and can be of several data types. To give the value 1 to object *a*, simply enter equivalently:

```
    a <- 1
    a = 1
```

To print values on the console:

```
    a
    a*2
    print(a)          # a is an object
    print("a")        # ``a'' is a character
    exp(pi)
```

2.3 Operators

```
a = 3 ; b = 6
a <= b
a != b
(b-3==a) & (b>=a)
(b ==a) | (b>=a)
```

2.4 Functions

Functions are defined as follows:

```
myfunction <- function(par=2) { par^2} # defines the function myfunction
myfunction                             # prints the function myfunction
myfunction()    # execute myfunction with the default values of the parameters
myfunction(3)   # execute myfunction for the value "3" of the parameter
```

It is a (very) good idea to write your function in a file *myfunction.R*. Function are loaded by sourcing the file or running the full script.

3 Data types

3.1 Variables

Vectors: the simplest data structure.

```
x = 1:10
x
x2 <- c(3,5,2,10)
x2
length(x2)
y = 2*x + 3
y[5] ; y[1:3] ; y[-3]
```

Lists: lists are a general form of vector in which the various elements need not be of the same type, and are often themselves vectors or lists. Lists provide a convenient way to return the results of a statistical computation.

```
A = matrix(1:15,ncol=5)
x=list(mat=A, text="testliste",vec=y)
x[[2]] ; x$vec
```

Matrices: matrices or more generally arrays are multi-dimensional generalizations of vectors. In fact, they are vectors that can be indexed by two or more indices and will be printed in special ways.

```
A = matrix(1:15,ncol=5); A
B = matrix(1:15,nc=5,byrow=TRUE) ; B
A[1,3] ; A[,2] ; A[2,] ; A[1:3,1:3]
is.matrix(A)
as.list(A)
```

Arrays: an array can be considered as a multiply subscripted collection of data entries.

```
array(1:12,c(2,3,2))
array(c(1:8, rep(1,8),seq(0,1,len=8)), dim = c(2,4,3))
```

Data frames: data frames are matrix-like structures, in which the columns can be of different types. Think of data frames as 'data matrices' with one row per observational unit but with (possibly) both numerical and categorical variables. Many experiments are best described by data frames: the treatments are categorical but the response is numeric.

```
t = c(147, 132, 156, 167, 156, 140)
p = c( 50, 46, 47, 62, 58, 45)
s = c("M","F","F","M","M","F")
H = data.frame(t,p,s)
H
str(H)

data(USArrests)
str(USArrests)
summary(USArrests)
tmp <- USArrests
 attach(tmp)  # Beware: cannot modify the data!
   mean(Murder)
   Murder <- Murder/2
   mean(Murder)
 detach(tmp)
mean(tmp$Murder)
with(tmp,mean(Murder))
```

3.2 Indices

```
d = c(2,3,5,8,4,6); d
d[2]
d[2:3]
d[-3]
d[-(1:2)];d[-c(1,2)]
(1:length(d))[d>5]
which(d>5)
```

NA (Not Available) indicates a missing value

```
d[3]=NA
d
summary(d)
is.na(d)
any(is.na(d))
all(is.na(d))
help(NA)
```

3.3 Objects classes

They are: integer, double, numeric, character, factor, logical.

Factors provide compact ways to handle categorical data

```
class(a)
class(s[3])
class(c(TRUE,TRUE,FALSE))
str(s)
summary(s)
str(as.factor(s))
levels(s)
levels(as.factor(s))
summary(as.factor(s))
```

4 Import/export

```
read.table(file.choose(),stringsAsFactors = FALSE, header=T, quote="\"",
+ na.strings="?",skip=0,comment.char="#")
tab <- read.table(file=``myfile.txt'')

write.table(tab,file="myfile2.txt",quote=FALSE,sep=" ",na="?",row.names=F)
```

Beware:

```
save(a,file="a.R")
load("a.R",envir=newenvir)
```

The first command saves the object a in a (R) binary format file named a.R . There is a compression option.
The second command loads the object included in the (R) binary format named a.R . **Be careful that this replaces all existing objects with the same names in the current environment**. You should load the files in a new environment as indicated.

5 Graphics

R is very powerful for graphics. See for instance https://en.wikibooks.org/wiki/R_Programming/Graphics, the examples and the corresponding codes explained in a dedicated book https://www.stat.auckland.ac.nz/~paul/RG2e/. See also http://zoonek2.free.fr/UNIX/48_R/03.html and http://zoonek2.free.fr/UNIX/48_R/04.html for many detailed examples.

Note that in RStudio, if you get the error "Error in plot.new() : figure margins too large", try to increase the size of the plot window.

Useful commands especially for command line sessions:

```
      dev.set(n)  # pour désigner la  fenêtre graphique n
      dev.list()    # liste toutes les fenêtres graphiques disponibles
      dev.cur()   # affiche le numéro de la fenêtre courante
      dev.new() # crée une nouvelle fenêtre graphique
      dev.off(x)   # ferme la fenêtre graphique correspondante
      plot.new()# efface la fenetre courante (aussi frame())
```

```
demo(graphics)
```

Some examples of graphics:

```
x1 <- 1 ; x2 <-3 ; x3 <- 5 ; y1 <- 2 ; y2 <- 4; y3 <- 3
plot(x1,y1, xlim=range(x1,x2,x3), ylim=range(y1,y2,y3))
points(x2,y2, col='blue')
points(x3,y3, col='red')
lines(c(x1,y1),c(x2,y2))
abline(1,c(1,2))

x = runif(50, 0, 2)
y = runif(50, 0, 2)
plot(x, y, main = "Titre", xlab = "abscissa", ylab = "ordinate", col = "darkred")
abline(h = .6, v = .6)
text(.6, .6, "comment on  the graph")
colors()
```

```r
hist.norm = function(n, col)
{
x = rnorm(n)
h = hist(x, plot = F)
s = sd(x)
m = mean(x)
ylim = c(0, 1.2 * max(max(h$density), 1/(s * sqrt(2 * pi))))
xlab = "Histogram and normal approximation"
ylab = ""
main = paste("Gaussian sample : n=", n)
hist(x, freq = F, ylim = ylim, xlab = xlab, ylab = ylab, col = col, main = main)
curve(dnorm(x, m, s), add = T, lwd = 2)
}

op = par(mfcol = c(1, 3))
hist.norm(200, col = "yellow")
hist.norm(800, col = "darkgoldenrod")
hist.norm(3200, col = "blue")
par(op)

x = rnorm(1000)
h = hist(x, freq = F)
lines(density(x))


# op <- par(mfrow=c(5,1),mar=c(3,3,1,1),mgp=c(1.5,0.5,0),oma=c(3,0,2,0))

x = rnorm(100)
op <- par(mfcol = c(2, 2), bg = "lightcyan")
boxplot(x)
boxplot(x, horizontal = T)
boxplot(x, col = "red")
boxplot(x, col = "orange", border = "darkblue")
par(op)
```