Introduction to R

Didier Fraix-Burnet

Statistics for Astrophysics: Clustering and Classification
D. Fraix-Burnet & S. Girard
EAS Publications Series, Vol. 77, 2016

INTRODUCTION TO R

Didier Fraix-Burnet [1, 2]

Abstract. I present a small introduction to R useful for the School. You need to execute and play with the commands.

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[1] for further information.

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

R is used by most statisticians in the world mainly for research and development. There are more and more packages for astronomical purposes, with a package to manage FITS format: FITSio.

CRAN[2] (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[3]. Simply asking your question in your favorite internet search engine, you will surely find the corresponding

[1] Univ. Grenoble Alpes, IPAG, F-38000 Grenoble, France
[2] CNRS, IPAG, F-38000 Grenoble, France
[1] http://www.r-project.org/
[2] https://cran.r-project.org/
[3] http://cran.r-project.org/manuals.html
http://www.statmethods.net/
http://cran.r-project.org/doc/manuals/R-intro.html

http://dx.doi.org/10.1051/eas/1677002

help and even full scripts. To make a search on the web regarding R, type "r-cran" instead of "R"! For MATLAB users, there are tables of command equivalence[4].

You can find a Short Reference Card[5] on the School website with the Codes and Data[6].

1.1 Installation of R

You can download a binary version for your platform. 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='YOUR PROXY ADDRESS: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 Graphical User Interface (GUI) or Integrated Development Environment

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

I would strongly recommend RStudio[7]. Otherwise there are many others, one being included in the Mac binary.

1.3 Packages

Packages are ensembles of programs, commands and scripts. They must be installed, i.e. compiled, once. 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 of mirrors) or from a previously loaded file.

To install packages:

[4] `http://mathesaurus.sourceforge.net/octave-r.html`

[5] `https://cran.r-project.org/doc/contrib/Short-refcard.pdf`

[6] The *R* codes for this chapter and the required data files can be found at `http://stat4astro2015.sciencesconf.org/`

[7] `http://www.rstudio.org/`

```
install.packages("fpc")
```

To load packages into memory and access its commands:

```
library(fpc)
```

To remove a package from memory:

```
detach(fpc)
```

Sometimes it is useful to update packages:

```
update.packages(repos="http://cran.univ-lyon1.fr/",ask=F,
                                                checkBuilt=T)
```

1.4 Files

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

- *.Rhistory*: 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. You can move to the right directory before loading R, or change directory within R:

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

RStudio encourages you to create projects so that it is very easy to handle several works and switch from one to the other.

2 Basics

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 unloaded packages
```

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

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).

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" requiring one parameter "par"
              # which is given the default value "2"
myfunction    # prints the function myfunction
myfunction()  # execute myfunction with the default values
              # of the parameters
myfunction(3) # execute myfunction for the value "3"
```

It is a (very) good idea to write a complex function into a file *myfunction.R*. It can be loaded by sourcing the file:

```
source(``myfunction.R'')
```

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="testlist",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

It is important to understand the indices since it can become somewhat tricky for lists or complex data frames. The following examples should help you in this respect.

```
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)
c1 <- c(1,2,3)
c2 <- c("a","b")
c <- list(c1,c2,100)
c
c[2]
```

```
c[[2]]
c[[2]][2]
c[[3]]
c[3]
c[[3]]+1
c[3]+1
```

The entry "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)
```

Some functions can handle these missing values in different ways (ignore or replace).

3.3 Objects classes

Objects can be of different classes: 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 Graphics

4.1 Introduction

R is very powerful for graphics[8].

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. Here are some useful commands especially for command line sessions:

[8] https://en.wikibooks.org/wiki/R_Programming/Graphics
https://www.stat.auckland.ac.nz/~paul/RG2e/
http://zoonek2.free.fr/UNIX/48_R/03.html
http://zoonek2.free.fr/UNIX/48_R/04.html

```
dev.new()  # opens a new graphical window
x11()      # also works
dev.list() # list all available graphical windows
dev.set(n) # to activate the windows "n"
dev.cur()  # gives the number of the current active window
dev.off(n) # closes window "n"
plot.new() # erase current window (also frame())
```

4.2 An illustration with simple classification tools

As an illustration of both the graphical capabilities of R and some basic classification techniques using kmeans and DBSCAN, we use the built-in database iris.

```
?kmeans
kmeans
colnames(iris)
iris2 <- iris[,-5]
colnames(iris2)

checkkmeans <- function(nstart=1000){
iris.kmeans <- kmeans(iris2,3,nstart=nstart)
plot(iris2[c("Sepal.Length","Sepal.Width")],
                              col=iris.kmeans$cluster)
points(iris.kmeans$centers[,c("Sepal.Length","Sepal.Width")]
                                        ,col=1:3,pch="*",cex=5)
                        }

checkkmeans(1)    # repeat the command many times and
                                          compare the results
checkkmeans(10)  # idem. Is it more stable?
iris.kmeans <- kmeans(iris2,3,nstart=1000)
```

Now let's perform a classification using DBSCAN:

```
library(fpc)
iris.ds <- dbscan(iris2,eps=0.42,MinPts=5)
table(iris$Species,iris.ds$cluster)
plotcluster(iris2,iris.ds$cluster)
detach(fpc)
```

Since there is generally not one best method, it is always a good idea to compare the results obtained using several approaches. Here is the comparison between kmeans and DBSCAN of the above analyses through a convenient graphical display:

```
table(iris$Species,iris.kmeans$cluster)
table(iris.kmeans$cluster,iris.ds$cluster)
op <- par(mfrow=c(2,1))
```

```
plot(iris2[c("Sepal.Length","Sepal.Width")],
                              col=iris.kmeans$cluster)
points(iris.kmeans$centers[,c("Sepal.Length","Sepal.Width")]
                                          ,col=1:3,pch="*",cex=5)
plot(iris2[c("Sepal.Length","Sepal.Width")],
                                              col=iris.ds$cluster)
par(op)
```

4.3 Other graphical examples

Here are other 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()

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))

x = rnorm(100)
op <- par(mfrow=c(2,2),mar=c(3,3,1,1),mgp=c(1.5,0.5,0),
      oma=c(3,0,2,0)) # to play with margins between plots
boxplot(x)
boxplot(x, horizontal = T)
boxplot(x, col = "red")
boxplot(x, col = "orange", border = "darkblue")
par(op)

demo(graphics)
```

5 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(x,file="whatever.RData")
load("whatever.RData",envir=newenvir)
```

The first command saves the object x in a (R) binary format file named whatever.RData. There is a compression option.
The second command loads the object included in the (R) binary format named whatever.RData. **Be careful because this replaces all existing objects with the same names (here object x) in the current environment**. To be safe, if you are not sure, you should load the files in a new environment as indicated. In this case two objects with the same name can exist, each in a different environment. To define the new environment and use it, this is rather simple:

```
newenvir <- new.env()
ls(newenvir)
newenvir$x
attach(newenvir)
x
detach(newenvir)
rm(x,envir=newenvir)
```