The R Language

B. Ghattas [1]

Aix-Marseille University

1. badih.ghattas@univ-amu.fr

Outline

- The R environment
- Basic Objects
  - Vectors
  - Matrices
  - Dataframes
  - Lists
- Programming with R
- Graphics
- Interaction with the system and with foreign languages :

  C,C++, fortran, Java, HTML, LaTeX
- Building packages

The R software

- Free
- Available and Compatible on all platforms
- Object oriented language
- Based upon packages (several thoushands actually available)
- Sources available for compilation

`http://lib.stat.cmu.edu/R/CRAN/`

To begin..

- RGUI.exe, RTERM.exe are in the bin directory
- q() to quit
- Esc touch to stop a program
- Up and down arrows navigate through the commands history
- Save created objects and actual session, File,$--> $ Save Image
- help(a function name)...
- Several instructions per lign must be separated by ;
- To see what an object is, type its name folllowed by Enter
- Reserved words : `NA` , `letters`

First examples

- Everything within R is an object. Different affectation operators are available : = ou <- ou _.
  Examples :
  ```
  n = 5
  m = c(1,2,3)
  k =c("a","b","c")
  n2 = c(T,F)
  ```
- R is case sensitive : `n` is different from `N`
- Managing objects :
  ```
  objects()
  ls(pat = "n")
  rm()
  remove()
  ```
- Some objects are automatically generated : `last.warnings`
- `c()`, `objects()` et `ls()` are functions.

Using Scripts

They gother several R instructions and functions. When only one instruction appears on a line, semicolumns are not needed.

- Use any editor : Emacs, Notepad++, or R's integrated editor.
- To Run a script use the menu or :

  `source("a:\\monprog.R")`
- To avoid executing a line, it should begin with : #
- To print a text or an object use

  `print()`
- A script may call other scripts using

  `source()`

Generating vectors

A vector may be generated in different ways :

- Create it giving the different values as arguments separated by comas to the function `c()`.
- Generate series using the operator `:`, the functions `seq()`, `rep()` or random numbers generators functions such as `sample()`, `rnorm()`, `runif()`.
- Use binary numerical operators (`+,-,*,/,^`) or logical operators(`&,|,!,>,<,>=,<=,==,!=`)
  Exemples : v1 and v2, are 2 vectors having the same length :

```
v3 = v1 + v2
v3= v1 / v2
v3 = v1^2
v3 = v1 > v2
v4 = 2:12
```

- `length(v)` returns the length of vector `v`.

Subsetting from vectors

- 2 equivalent ways to subset from vectors :
  - Use position index of the desired elements of `v`
  - Use a logical vector of the same length as `v` having TRUE at the desired positions.

  Examples :

```
v =-3:2
v[3]
v[c(4,6)]
a= c(4,6)
v[a]
a= c(T,T,F,F,T,T)
v[a]
a= v>0
v[a]
a = which(v<0)
v[a]
```

- Using negatif index to unselect elements `w = v[-4]`

Vectors mode, special operations

- numeric, character, logical, factor
- `factor()`, `cut()` creates factors.
- The mode of a vector may be tested or modified : `as.numeric()`, `is.numeric()`, `as.character()`, `is.character()`
- Each mode has specific operators. You can't sum vectors having the mode character.
- Some operations are possible although strange for beginner : `z=v+2` ,`z=v+w` (where `v` and `w` do not have necessarily the same length)

Random numbers generating

- For each distribution there exist in general at least four functions whose name is the distribution name prefixed by one of the following letters "d","p", "q","r" doing the following actions :
  - "d" : giving the density value $f(x)$ for a continuous distribution, or the probability value $P[X = x]$ otherwise.
  - "p" : computes the distribution function at point x, thus returns $P[X \leq x]$ (cumulative distribution function)
  - "q" : for a fixed q, returns the corresponding quantile x such that $P[X \leq x] = q$
  - "r" : gives a random number..
- Examples : rnorm, runif, rpois,rbinom
- Sample may be used for uniform and non uniform discrete distribution, but also for sampling with or without replacement.
- To analyze the generated vectors the following functions may be used : summary(), table(),hist(), quantile(), fivenum(), stem(), density(), rug(), qqplot()

Matrices

- To create a Matrix use

  `M=matrix(v,nrow=,ncol=)`

- Subsetting from matrices

  `M1 = M[1,2]; M2=M[c(2,3),c(3,4)]; M3 = M[a,b]`
  `M4= M[i,]; M5=M[,j]; M6=M[c(k,l),]`
  `M7= M[,c(k,l)]; M8=M[v,]; M9= M[,v]`

- M[v] linear index (per column), M[-v]
- Subsetting using logical operations : `M[M[,1]>0, ]`
- Reshaping a matrix. If `M` is a 6×2 matrix then
  `dim(M) = c(3,4)`
  converts it into a matrix 6×2.

Operations over matrices

- The classical binary operators are element-wise.
- %*% for the algebraic product
- To transpose a matrix use : `t()`
- To extract the diagonal of a matrix or create a diagonal matrix : `diag(v)`, `diag(M)`
- `colMeans` computes the means of each column. See also `rowMeans`,`colSums`,`rowSums`.
- To apply any arithmetic function (even yours) to any dimension of a matrix use : `apply()`
- Add a number or a vector to a matrix : `M+2`, `M+v`

Generating lists

- Creating unnamed lists

```
ll =list()
ll = list(1:10,letters)
```

- named list

```
ll =list(vect=1:10,alphab=letters)
```

- Lists as a result returned by some functions.

  Example : the function `split()`

```
v1 = letters
v2 = sample(1:2,26,replace=T)
ll  = split(v1,v2)
```

Subslecting from lists

Consider the list :

```
ll =list(vect=1:10,alphab=letters,mat = matrix(rnorm(100),ncol=2)
```

- `ll[[1]]` : the result is an object having the same nature as the first element of the list, a vector in this case. Only one element may be selected by this way.
- You may use also the $ operator : `M = ll$mat`
- `ll[1]`, `ll[c(2,5)]` : the result is a list, several elements may be extracted at once.
- The function `lapply()` applies any function to each component of a list. Example :

```
scores = sample(1:20,50,replace=T) #generates scores
sex = sample(c("F","M"),50,replace=T) #generates sex
mylist = split(scores,sex)
l2=lapply(mylist,mean)
```

`l2` is a named list having 2 elements (as much as there are differents valued in sex).

Dataframes

- Internaly they are lists (the same class). In appearance, they look like matrices.
- Columns within a dataframe may be of different modes
- Lines and columns are named and mau be accessed either bu their name or by their position.
- Example : If DD is a dataframe where the fifth columns is named "age", you can access the age data with :

```
DD[,5] or DD[,"age"] or # like for dataframes
DD$age or DD[[5]] or DD[["age"]] # like for lists
```

- You may use `rownames()` to modify row names for a dataframe. See also `colnames()` and `dimnames()`
- This is the default class for imported data sets.

Read and write tables

- `read.table(file=, sep=, header=)`
- `write.table(x=,append=, col.names=, row.names=, quote=)`

You may import other files coming from other software : Excel, SPSS, SAS, matlab,.... Use for that specialized function found for example in the package `foreign`.
dump(), source(),save(),load()

Loops and tests

- Tests : `if(expr1) expr2 else expr3`
  or vectorized version : `ifelse (condition,a,b)`
  Example :

```
x = c(6:-4)
sqrt(x)  # gives warning
  sqrt(ifelse(x >= 0, x, NA)) # ok
```

- Loops

```
for( nom in expr1) expr2
repeat expr2          (use break to stop)
while (condition) expr2
```

- You can use `next` to skip an iteration.

Writing your own functions

```
nom = function(arg1,arg2,arg3,...) expression
```

expression may be :

```
{
expr1;
expr2;
...;
}
```

Use `fix` or `edit` to modify a function.

Operator as function

```
"%!%" = function(x,y) {expression}"'
```

Use : `a %!% b`

Arguments order

Consider a function you have written :
`fun1 = function(a,b,c,d) { ...}`
The following calls are equivalent :

```
ans = fun1(a0,b0,c0,d0)
Ans = fun1(a0,b0,d=d0,c=c0)
ans = fun1(c=c0,a=a0,d=d0,b=b0)
```

Arguments with default values. `fun1 = function(a,b,c=c1,d=d1) {...}`
Possible calls :

```
ans = fun1(a0,b0)
ans = fun1(a0,b0,c=c0)
```

The argument `...`

```
 fun1 = function(a,b,c=c1,d=d1,...)
{ # c is logical
expressions
if ( c )  par(pch="*",...)
 }
```

Dealing with windows and panels

Several windows possible, only one active. `dev.list()` : list of available windows (by number)
`dev.set()` : Activates one window
Partionning windows
`split.screen()` Ex : split.screen(c(1,2))
`screen()` selects
`erase.screen()` deletes
`layout()` Ex : layout(matrix(1 :4,2,2))
Plotting goes automatically from one section to the other. Can have odd number of sections. You can give dimensions to each section.
`layout.show()` shows the partition of your window.
`par(mfrow=c(2,2))` same thing with equal dimensions for the sections.

Some graphical functions

- Scatterplots :

  `plot(x)` : x values against their order
  `plot(x,y)` : scatterplot
  `pairs()` : pairs of scatterplots.
  `matplot()` : plotting pairs of columns
- Distribution representation :

  `sunflowerplot()` :
  `piechart()` :
  `boxplot()` :
  `hist()` : histogram.
- Other graphics :

  `coplot()` : conditional scatterplots.
  `barplot()` : Barplots
  `qqplot()` : Quantile quantile plots
  `persp()` : 3D en perspective.
- Controling graphical parameters : `par()`

Some useful parameters

```
add =T : to
axes =T : adding or omitting axes
type = "p" ou "l" .. :

xlab= :
ylab= :

main= : main title
sub=  : subtitle
```

Some useful parameters

points(), lines(), text(), abline()
segments(), arrows(), rect(), polygon() legend(),title(), locator()
N.B. With text() we can display equations in TEX format using expressions.
Saving in postscript format : Use the menu, or right click, or use a functions :

```
postscript("d:\\myfig.eps")
plot(sin)
dev.off()
```

Interface with the system and other languages

- Call system comands : `system()`
- Produce latex represetantions of tables : use function `latex` from the package `Hmisc`
- Produce HTML represetantions of tables and graphics : use function `html` from the package `Hmisc`
- Use C or fortran code : You have to :
  - Compile your code, create object and dll (or so) files using the add ons programs Rtools.exe (for windows)
  - Dynamically load your code within R using `dyn.load` or `dynamic.library()`
  - Write a R interface for your C or fortran functions