Université Paris Dauphine 2012-2013
U.F.R. Mathématiques de la Décision
MAT-6-3: Exploratory statistics **(NOISE)**

Initiation to R

Jean-Michel MARIN, Robin RYDER

Contents

1 Introduction

R is an interactive programming language. It is interpreted and object-oriented and contains a very large collection of statistical methods as well as important graphical facilities. It is a free clone of S-Plus (software sold by MathSoft based on language S).

It was initiated in the 1990s by Robert Gentleman and Ross Ihaka. Many contributors have joined them since. The website of the "R core-development Team", `http://www.r-project.org`, is the best source of information on R. You will find there many distributions of R, as well as packages and help files. Packages are also available from the "comprehensive R archive network" (CRAN) `http://cran.r-project.org/`.

2 Using R under Windows

We suggest you use the IDE RStudio (`http://www.rstudio.org`), which offers an easy-to-use Graphical User Interface.

On Unix systems, R can be used directly from the command line, by entering the command `R` in a terminal. You can also use the IDE RKward.

In most IDEs, you will find several windows. The most important ones are the script window and the console window. In the console, you can type commands which are executed on the go. In the script window, you can type more complex scripts, which are executed in the console when you press the Execution button.

When using the command line in Unix, you only have access to the console. You can write scripts in a text editor, and then call them in the console using the command `source("filename.R")`.

3 Objects

The basic elements of R are objects which can be data (vectors, matrices, time series...), functions, graphics...

An R object is defined by its type, which describes the contents, and its class, which describes the structure.

The main types are:

```
null (empty object), logical, numeric, complex, character
```

The main classes are:

```
vector, matrix, array, factor, time-series, data.frame, list
```

A vector, matrix, table, factor or time series can be of type `null (empty object), logical, numeric, complex, character`, but a list or a data frame can be heterogeneous and include different types.

3.1 Vectors

Vectors are the basic type in R. A vector is an entity made of an ordered collection of elements of same nature. In R, vectors can be made of numerical, logical or alphanumerical elements.

To build manually a vector, use the function `c(item1,item2,...)`. Decimal numbers must be encoded with a decimal point, character strings must be in double quotes `" "`, and logical values are encoded by the strings `TRUE` and `FALSE` or their abbreviations `T` and `F`. Missing values are encoded by the character string `NA`.

The console starts with the character `>`. This signals that the console is available for you to type an instruction. When you type an unfinished instruction (for example if you open a parenthesis and do not close it before pressing Enter), the console line starts with the character `+` instead, indicating that it is waiting for the end of the instruction.

In the console, you are now going to type the following commands; you do not need to type the character `>`.

`> a=c(3,5.6,1,4,-7)`	Create object `a` containing a numerical vector of length 5 and with the values 3, 5.6, 1 ,4, -7.
`> a`	Display vector `a`
`> a[1]`	Display the first value of vector `a`
`> b=a[2:4]`	Create numerical vector `b` of length 3 and with values 5.6, 1, 4
`> b`	
`> d=a[1,3,5]`	Error message
`> d=a[c(1,3,5)]`	Create numerical vector `d` of length 3 and with values 3,1,-5
`> d`	
`> 2*a`	Multiply by 2 each value in vector `a` and display the result
`> e=3/d`	Create numerical vector `e` of length 3 and with values 1, 3, -3/7
`> e`	
`> f=a-4`	Create vector `f` of length 5 and whose values are equal to those of `a` minus 4
`> d=d+e`	Replace vector `d` with the vector made from adding vectors `d` and `e`
`> d=d-e`	Replace vector `d` with the vector made from the difference between vectors `d` and `e`
`> d*e`	Element-wise multiplication of vectors `d` and `e`
`> e/d`	Element-wise division of vectors `e` and `d`
`> sum(d)`	Compute the sum of `d`
`> length(d)`	Length (number of elements) of `d`
`> t(d)`	Transpose vector `d`; the result is a row vector
`> t(d)%*%e`	Scalar product of row vector `t(d)` and column vector `e`
`> g=c(sqrt(2),log(10))`	Create numerical vector `g` of length 2 and with values $\sqrt{2}, \log(10)$
`> (1+exp(2))/cos(8)`	R is also a calculator
`> bool=d==5`	Create boolean vector `bool` of length 3 with value `TRUE` if `d[i]=5` and `FALSE` otherwise
`> text=c("big","small")`	Create character vector `text` of length 2 with values `big`, `small`
`> is.vector(d)`	Call function `is.vector()` which returns the logical value `TRUE` if its argument is a vector and `FALSE` otherwise

Some useful functions:

```
> rep(1:4,2)
> rep(c(1.4,3,5),each=2)
> ?rep                          Access the help page for the function rep()
> seq(0,1, length=11)
> seq(1.575, 5.125, by=0.05)
> help(seq)
> sort(b)
> sample(c(0,1),10,rep=T)
> sample(c(1,2,3),3)
> letters[2]
> LETTERS[2]
> sample(letters[1:9],5)
> order(d)
> help(order)
> help.start()
> abs(d)
> log(abs(d))
```

3.2 Matrices

Like vectors, matrices can be of any type, but cannot contain elements of different types. The syntax to create an $n \times p$ matrix is: `matrix(vec,nrow=n,ncol=p,byrow=T)` where `vec` is a vector containing the elements of the matrix, which will be filled by column (unloss the option `byrow=T` is included).

```
> a=1:20
> b=sample(1:10,10)
> x1=matrix(a,nrow=5)
```
Create numerical matrix `x1` of dimension 5×4 with first line 1,6,11,16

```
> x2=matrix(a,nrow=5,byrow=T)
```
Create numerical matrix `x2` of dimension 5×4 with first row 1,2,3,4

```
> x3=t(x2)
```
Transpose matrix `x2`

```
> x4=matrix(b,ncol=2)
> x1;x2;x3;x4
> b=x3%*%x2
```
Matrix product between `x2` and `x3` (R controles for dimension adequacy)

```
> f=x2%*%x4
> b
> f
> dim(x1)
```
Display the dimension of matrix `x1`

```
> dim(x4)
> b[3,2]
```
Select element `[3,2]` of matrix `b`

```
> b[,2]
```
Select second column of `b`

```
> b[c(3,4),]
```
Select third and fourth rows of `b`

```
> b[-2,]
```
Remove second row of `b`

```
> b[,-c(2,4)]
```
Remove second and fourth columns of `b`

```
> a=sample(1:10,30,rep=T)
> a=matrix(a,nrow=6)
> a
> a>5
```
Display a matrix of booleans where element `[i,j]` is `TRUE` if `a[i,j]>5`

```
> a[a<5]=0
```
Set to 0 all elements of `a` lesser than 5

```
> rbind(x1,x2)
```
Vertical concatenation of matrices `x1` and `x2`

```
> cbind(x1,x4)
```
Horizontal concatenation of matrices `x1` and `x4`

```
> apply(x1,2,sum)
```
Compute the sum of `x1` column-wise

```
> apply(x1,1,sum)
```
Compute the sum of `x1` row-wise

3.3 Matrices of more than two dimensions

Matrices of more than two dimensions are created with the command: `array(vec,c(n,p,q...))` where `vec` is a vector containing the elements of the matrix, which will be filled by column, and the argument `c(n,p,q...)` gives the dimensions: `n` is the number of rows, `p` the number of columns, `q` the number of matrices...

```
> x=array(1:50,c(2,5,5))
> x
> x[1,2,2]
> dim(x)
> aperm(x)   Generalized transpose of x: x[i,j,k] becomes x[k,j,i]
```

3.4 Lists

A list is an ordered collection of objects, which can be of different types. The elements of a list can thus be any objects defined in R. This property is used by some functions to return complex results as a single object.

Building a list is done with the function `list(name1=item1,name2=item2,...)`, where the names are optional. Each element of a list can be accessed either with its index in double square brackets `[[...]]`, or with its name preceded by the symbol `$`.

```
> li=list(num=1:5,y="colour",a=T)
> li
> li$num
> li$a
> li[[1]]
> li[[3]]
> a=matrix(c(6,2,0,2,6,0,0,0,36),nrow=3)
> eigen(a,symmetric=T)                                  Diagonalize a
> res=eigen(a,symmetric=T)
> res$values
> res$vectors
> a
> diag(res$values)
> res$vectors%*%diag(res$values)%*%t(res$vectors)
```

3.5 Data frames

In R, a data frame is similar to a matrix, but the columns can be of different types. Data frames constitute a special class of lists, devoted to storing data for analysis. Each component of the list is the equivalent of a column or variable, and the elements of these components correspond to rows or individuals. Specific methods are associated with this class.

To create a data frame, group variables of same length with the command `data.frame(name1=var1, name2=var2,...)`. It is also possible to transform a matrix into a data frame with the command `as.data.frame(mat)`. Finally, it is possible to use an external file.

`> v1=sample(1:12,30,rep=T)`	Sample with replacement from integers 1 to 12
`> v2=sample(LETTERS[1:10],30,rep=T)`	
`> v3=runif(30)`	30 independent realisations of the uniform distribution over [0,1] (see section 4)
`> v4=rnorm(30)`	30 independent realisations of the normal distribution with mean 0 and variance 1
`> v1;v2;v3;v4`	
`> xx=data.frame(v1,v2,v3,v4)`	
`> xx`	
`> xx$v2`	
`> summary(xx)`	
`> xx=data.frame(v1,factor(v2),v3,v4)`	Build the data frame `xx` with variable `x2` declared as a factor (qualitative variable)
`> summary(xx)`	
`> ma=matrix(1:15,nrow=3);ma`	
`> plot(ma)`	
`> ma=as.data.frame(ma)`	
`> is.data.frame(ma)`	
`> plot(ma)`	
`> data()`	Open a window listing all data frames available in R
`> data(women)`	Load data frame `women`
`> names(women)`	Display variable names of data frame `women`
`> attach(women)`	
`> height`	

4 Usual distributions

Distribution	Name	Parameters	Default values
Beta	`beta`	`shape1, shape2`	
Binomial	`binom`	`size, prob`	
Cauchy	`cauchy`	`location, scale`	0, 1
Chi-Squared	`chisq`	`df`	
Exponential	`exp`	`1/mean`	1
Fisher	`f`	`df1, df2`	
Gamma	`gamma`	`shape,1/scale`	-, 1
Geometric	`geom`	`prob`	
Hypergeometric	`hyper`	`m, n, k`	
Log-Normal	`lnorm`	`mean, sd`	0, 1
Logistic	`logis`	`location, scale`	0, 1
Normal	`norm`	`mean, sd`	0, 1
Poisson	`pois`	`lambda`	
Student	`t`	`df`	
Uniform	`unif`	`min, max`	0, 1
Weibull	`weibull`	`shape`	

For each distribution, there are four commands, prefixed by the letters `d, p, q, r` and followed with the name of the distribution. For example, for the normal distribution: `dnorm, pnorm, qnorm, rnorm` or for the Weibull distribution: `dweibull, pweibull, qweibull, rweibull`.

`dname`: this is the probability density function for a continuous variable, and the probability mass function ($\mathbb{P}(X = k)$) for a discrete distribution.

`pname`: this is the cumulative distribution function ($\mathbb{P}(X \leq x)$) ;

`qname`: this is the quantile function, in other words the value at which the cumulative distribution function reaches a certain probability. In the continuous case, if `pnorm(x)=y` then `qnorm(y)=x`. In the discrete case, it returns the smallest integer u such that $F(u) \geq y$ where F is the relevant cumulative distribution function.

`rname`: generates independent random realisations from the distribution.

```
> qnorm(0.975);dnorm(0);pnorm(1.96)
> rnorm(20);rnorm(10,mean=5,sd=0.5)
> x=seq(-3,3,0.1);pdf=dnorm(x)
> plot(x,pdf,type="l")
> runif(3)
> rt(5,10)
```

5 Some useful functions for exploratory statistics

```
> data(women)
> names(women)
> attach(women)
> mean(height)                          Empirical mean of quantitative variable height
> var(height)                           Empirical variance of height
                                        unbiased estimator (denominator n − 1)
> sd(height)                            Standard deviation of height
> median(height)                        Empirical median of height
> quantile(height)                      Empirical quantiles of height
> summary(weight)                       Summary of weight
> summary(women)                        Summary of women
> hist(weight,nclass=15)                Histogram of weight with15 classes
> boxplot(weight)                       Box plot ofweight
> cor(height,weight)                    Empirical linear correlation coefficient
                                        between weight and height
> v1=rnorm(100)
> hist(v1)
> v2=factor(sample(letters[1:4],100,rep=T))
> table(v2)                             Summary of qualitative variable v2
> barplot(table(v2))                    Bar plot of v2
> pie(table(v2))                        Pie chart of v2
> boxplot(v1~v2)                        Box plot of v1 for each modality of v2
```

6 Plots in R

Plots are useful to visualize your results and to check that you have obtained what was required. Learn to make them! It is important that you use the correct options: the default values can sometimes lead to the wrong plot!

Preliminary remarks

- `par(mfrow=c(nrow,ncol))` is used to display several plots side by side.
- To save a plot as an .eps file, use the command `dev.copy2eps(file="plotname.eps")` after creating the plot.
- Always start with a histogram [`hist()`] or a scatter plot [`plot()`], to which you can then add points, curves or straight lines.

6.1 Histograms

Histograms are mostly used to visualize a distribution.

The syntax is `hist(x,options)`. The following options will be useful:

- `breaks = nb`: to specify the number of classes (of "bars").
- `probability = TRUE`: to display a histogram in probability, rather than in counts (which is the default).

Observe the difference between the two following plots:

```
> attach(women)
> hist(height, breaks=15)
> hist(heght, breaks=15, probability=TRUE)
```

Also, note that when you create a new plot, the previous plot disappears.

6.2 Scatter plots

Scatter plots are useful to visualize the link between 2 variables.

The syntax is `plot(x,y, options)`. Note that `plot(a,options)` puts the variable `a` on the y-axis and the indices on the x-axis.

You can choose to display only points, which is the default and corresponds to `type="p"`, (`pch=nb` to change the symbol, `cex=nb` for its size), or `type="l"` to have lines between the points (`lwd = nb` to change its width) or `type="b"` to have both.

6.3 Options common to histograms and scatter plots

The following options are used to add titles and captions to your plots:

- `col = "red"`: to change the colour (use quotation marks).
- `main = ("title")`: to add a title to the plot (displayed above).
- `xlim = c(xmin,xmax)`: to choose the range of the x-axis (use `ylim` for the y-axis).
- `xlab = ("x caption")`: to change the caption on the x-axis (`ylab` for the y-axis).

6.4 Additions to an existing plot

- **Points:** `points(x,y)` uses the same principle as `plot(, type="p")`.
- **Lines:** `lines(x,y)` where `x` and `y` are the coordinates of points to link; uses the same principle as `plot()` with option `type="l"`.
- **Horizontal and vertical lines:** `abline(v=nb)` displays a vertical line of equation $y = nb$, `abline(h=nb)` a horizontal line. The available options are the same as `plot()` with option `type="l"`.
- **Curves:** `curve(fonction, add=TRUE)` adds a curve to an existing plot. Without `add=TRUE` a new plot is created.

6.5 Examples

`> x=rnorm(100)`	100 draws of the $N(0,1)$ distribution
`> hist(x,breaks=20, probability=TRUE)`	Create a histogram
`> curve(dnorm(x),add=TRUE,col="red")`	Add a curve to the plot
`> curve(cos(x), xlim=c(-10,10))`	Without `add=TRUE`, a new plot is created
`> t=seq(-10,10,by=0.1)`	
`> points(t, sin(t), pch="x")`	Add points to graph
`> plot(t, sin(t), pch="x")`	With `plot()`, a new plot is created. Note the default range of the x-axis.

7 Building a new function

You can build your own functions. For this, we suggest that you use the script window, or a text editor if you are using R in the command line.

Generally speaking, a new function is defined using the following expression:

```
function_name=function(arg1[=default1],arg2[=default2],...) {
block of instructions # comments after a hash sign
x=...
return(x)
}
```

The curly brackets signal the beginning and the end of the function source code. The square brackets are not part of the expression; they indicate that default values are not compulsory.

Upon execution, R returns the value given in the statement `return()`. If this statement was omitted, it returns the output of the last evaluation made inside the function. All

values assigned are local in scope. Upon execution, a copy of the arguments is sent to the function; the original values are not changed.

```
> y=1
> square=function(x) { y=x*x ; return(y)}
> square(2)
> y
> fix(square)
```

Include several statements on the same line by separating them with a semicolon `;`

Finally, note that it is possible and recommended to add comments inside the code of your functions, by using the hash symbol `#`. After this symbol, the rest of the line is ignored upon interpretation and can thus be used for any comments.

8 A few notions of programming

In this section, we briefly treat of the selection and looping commands `if, while, for`.

```
> bool=T
> i=0
> while(bool==T) {i=i+1; if (i>10) {bool=F}}
> i
> s=0
> x=rnorm(10000)
> for (i in 1:10000) {s=s+x[i]}
> s
> un=rep(1,10000)
> t(un)%*%x
> s=0
> system.time(for (i in 1:10000) {s=s+x[i]})[3]
> system.time(t(un)%*%x)[3]
```

R can encounter memory issues if you call a very high number of loop iterations, even if they contain very simple instructions. Indeed, as shown by the last two commands, loops cost a lot of processing time. As far as possible, you should therefore refrain from using loops and replace them with matrix operations (the matrix operators in R use loops written in C, which are much faster).

9 Handling created objects

The command `ls()` will list all objects you have created. The command `rm()` is used to delete objects.

`>a=1:10`	
`>b=list(10,"foo",a)`	
`>f=function(x){return(x/3)}`	
`>ls()`	
`>rm("f")`	
`>ls()`	The function f does not exist anymore
`>f(3)`	Error message
`>rm(list=ls())`	Delete all existing variables
`>ls()`	Empty workspace

You should save your scripts very regularly as files ending in the extension .R.

Note that upon exit, R also offers to save your objects (as a .RData file) and your console history (as a .Rhistory object).