Brief History of R

- S: language for data analysis developed at Bell Labs circa 1976
- Licensed by *AT&T/Lucent* to *Insightful Corp*. Product name: *S-plus*.
- R: initially written & released as an open source software by Ross Ihaka and Robert Gentleman at U Auckland during 90s (R plays on name "S")
- Since 1997: international R-core team ~15 people & 1000s of code writers and statisticians happy to share their libraries!

R is an "Open source" program (free software)

- You don't have to pay for it and it:
  - Provides full access to algorithms and their implementation
  - Gives you the ability to fix bugs and extend software
  - Provides a forum allowing researchers to explore and expand the methods used to analyze data

R is an interpreted computer language.

- R is used for data manipulation, statistics, and graphics. It is made up of:
  - operators (+ - <- * %*% ...) for calculations on arrays & matrices
  - large collection of built-in functions
  - user written functions & huge amount of functions (packages) written by contributors;
- Most functions are written in R itself, calling upon a smaller set of internal primitives.
- It is possible to interface procedures written in C, C+, or FORTRAN languages for efficiency, and to write additional primitives.

There are over 7200 add-on packages

(http://cran.r-project.org/src/contrib/PACKAGES.html)

This is an enormous advantage - new techniques available without delay, and they can be performed using the R language you already know **but** as the number of packages grows, it is becoming difficult to choose the best package for your needs.

Learning R

- Read through the CRAN website
- Use google or http://www.rseek.org/
- Many books (and pdf files)
- Lots of videos: http://www.youtube.com/watch?v=1jl9OKnjQHs

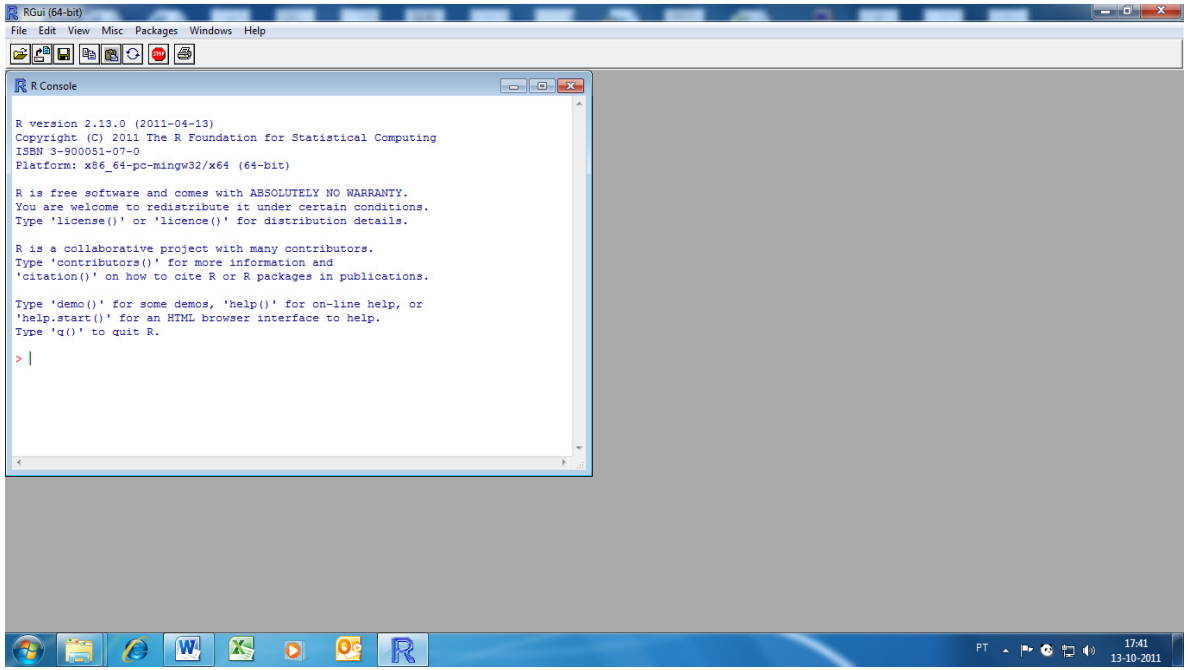
How to get R:

The Comprehensive R Archive Network (CRAN) at http://www.r-project.org or google it.

Download the executable file and it will install itself

How to start R:

Double click R icon on your desktop or use the start button

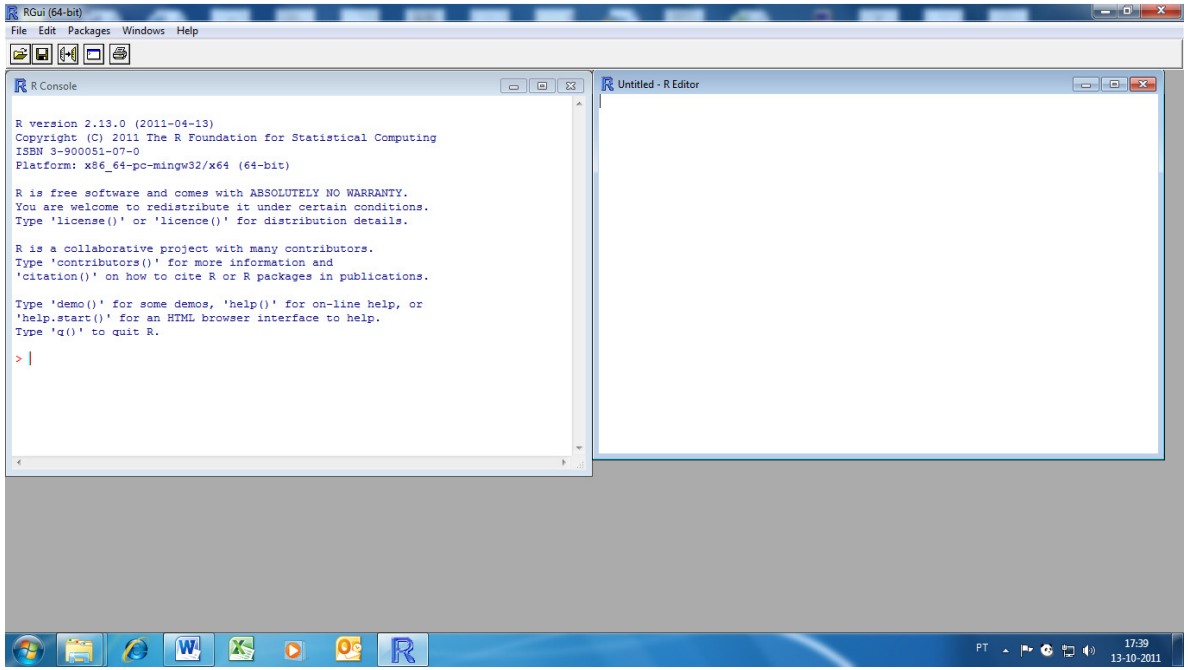


R is an interactive program: you type commands and you obtain answers

Examples

While one can type R commands one line at a time directly into the R console this is not at all efficient for writing programs. So instead most users type R commands into a text editor and then copy and paste them into R.

Menu: File / script file (new or open)


RGui (64-bit)
File Edit Packages Windows Help
R Console
R version 2.13.0 (2011-04-13)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-pc-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Untitled - R Editor
PT
17:39
13-10-2011

Some preliminaries on entering commands:

- Expressions and commands in R are case-sensitive.
- Command lines do not need to be separated by any special character but if you write two or more commands in the same line you must separate them using a semicolon

- Anything following the character (#) is ignored by R – considered as a comment.
- An object name must start with an alphabetical character, but may contain numeric characters thereafter. A period may also form part of the name of an object. For example, x.1 is a valid name for an object in R.
- You can use the arrow keys on the keyboard to scroll back to previous commands.

Arithmetic operators

+ - * /	As usual
^	Power 2^5 is 2^5 (same as **)
%/%	Integer division 20 %/% 3 =6
%%	Modulus (remaining of the integer division) 20 %% 3 =2

Usual priority rules apply. Parenthesis can be used.

Examples:

```
> 9+2
[1] 11
> 2+9*3^2
[1] 83
```

Some useful built-in numeric functions

Almost everything in **R** is done through functions. Here are some numeric functions that are commonly used.

abs(x)	absolute value
sqrt(x)	square root
log(x)	natural log of x (base e)
log10(x)	decimal log of x
exp(10)	exponential (e^x)
sin(x),cos(x) ...	sinus, cosinus,...

gamma(x)	gamma function $\Gamma(x)$ – generalized factorial
beta(x,y)	beta function $\beta(x,y)=\frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$ used in beta distribution
choose(n,x)	number of distinct x-elements subsets that can be chosen from n objects
factorial(x)	can be used with real values $(=\Gamma(x+1))$

For all these functions we can use their logarithm – lgamma(x) or lfactorial(x)

round(x,n=0) round to the nearest value with n decimal places (n=0 is the default)
trunc(x) truncated to the next integer toward 0
signif(x,n=0) round to the nearest value with n significant digits
Other rounding functions: ceiling(x), floor(x)

Built-in constant

pi π=3.14...

There are another built-in constants but related to dates (month.abb and month.name) or to characters (letters and LETTERS)

Exercises 1 –

1. Without using R replace the "?" by the correct answers

```
>3+2*5-4/2
[1]     ?
>3+2*(5-4)/2
[1]   ?
>round(6.321,1)-3*0.1
[1]     ?
>trunc(-6.7)^2
[1]   ?
```

2. Compute the roots of the equation $x^2 - 4x + 3 = 0$. Recall the formula $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

3. Let $f(x)$ be the density function of a standardized normal variable. Compute $f(0)$ and $f(2)$. Note that $f(x) = \left(1/\sqrt{2\pi}\right) e^{-x^2/2}$.

4. Let $X \sim b(10, 0.2)$. Compute $\Pr(X = 3)$. Note that $f(x) = \binom{10}{x} 0.2^x \, 0.8^{10-x}$

Assigning values to an object

<- -> =

Examples

n <- 10 or 10 -> n or n=10 (if you are using <- you can't separate "<" from "-")

R is case sensitive (N and n are different objects)

To print the content of an object:

print(n) or name the object n (works on most situations)

To give more than one command in the same line use the semicolon to separate them

Objects can be numbers, arrays (one or more dimensions), lists, …

Available functions for an objects depend on the object type (numeric, text, …)

Special commands: help(), ?,ls(), rm(), getwd(), setwd("…\rfiles") … → see R file

Logical operators (TRUE or FALSE)

< <= > >=	As usual
==	Equal (you need to write twice the symbol =)
!=	Not equal
x|y	x or y
x&y	x and y

Examples:

```
> 2<4
[1]  TRUE
> (2<5) & (7>4)
[1]  TRUE
```

As in most software TRUE is represented by 1 and FALSE by 0

```
> (2<5)+(4<7)
[1] 2
```

One dimensional arrays – Vectors

The basic data object in R is the vector (array with one dimension). Even scalars are vectors of length 1.

How to create a vector?

- Concatenation `x=c(1,2,5,9.3)`
  `y=c(1.3,4,2)`
  `z=c(x,y)`
- Using a pattern `x=seq(8,12,by=2)` creates the same as c(8,10,12)
  `y=seq(8,13,by=2)`
  `z=rep(4,3)` creates the same as c(4,4,4)
  `v=4:6` creates the same as c(4,5,6) or seq(4,6,by=1)

- You can mix both approaches `x=rep(1:3,4)` or `y=c(rep(2,3),9.1,4)`
- Function `scan()` less interrestimg

How to print a vector? Just name it or use the function `print()`

Arithmetic operations using vectors:

- Element by element (be careful! Vectors are re-arranged to allow operations)
  Examples: Compute v=x+y and w=x*y when x=c(2,7,9) and
  - y=c(1,5,7) element by element v=(3,12,16) w=(2,35,63)
  - y=c(1,5) y was extended by repeating the values as many time as needed
    y'=(1,5,1) and then element by element v=(3,12,10) w=(2,35,9)
  - y=2 v=(4,9,11) w=(4,14,18)

- Arithmetic functions can be used. Examples: x=c(2,7,9)
  v=sqrt(x) w=log(v) z=log(sqrt(x))

- Logical functions can also be used: x=c(2,7,9)
  > x>5
  [1] FALSE TRUE TRUE
  > x==9
  [1] ?
  > x=9
  [1] ?

How to select elements form a vector? x=c(2,5,9,1,6,4,5,3)

- Pick one element: x[3] →9 3rd element
- Pick a sequence of elements:
  x[c(1,5,7)] → (2 6 5) 1st,5th and 7th elements
  x[1:4] → ?
- Use a logical function to choose elements
  > y =x[x>3] selects all elements greater than 3 → y=(5 9 6 4 5)

How to delete elements form a vector? x=c(2,5,9,1,6,4,5,3)

Same approach but using a minus sign

> x[-3] deletes the 3rd element

> x[-(1:3)] deletes the first 3 elements

Some useful functions designed to operate with one dimensional arrays

sum()	cumsum()		
prod()	cumprod()		
min()	max()	median()	percentile()
mean()	var()	sd()	
length()			
rev()			
unique()	which()		

Exercises 2 –

1. Create the following vectors
   (1,2,3,…,20)
   (1,2,3,…,19,20,19,18,…,1)
   (4,4,4,7,7,7,7,7,7,10)
2. Create a vector of the values of e^x at x=3,3.1,3.2,…,6
3. Create a vector containing the first 50 odd numbers 1, 3, 5 …,99
4. Investigate if 23 is a prime number.
5. Calculate $\sum_{i=1}^{100}(i^3+4i^2)$
6. Let x=(27 82 115 126 155 161 243 294 340 384 457 680 855 877 974 1193 1340 1884 2558 15743).
   a. Calculate the sample mean and the sample standard deviation.
   b. How many observations are greater than the mean?
7. As you know, using Taylor expansion around a of the function f, we get
   $f(x)=f(a)+\sum_{k=1}^{\infty}\frac{(x-a)^k}{k!}f^{(k)}(a)$. Assume that $f(x)=e^x$. Compare the results for $f(0.5)$ and for $f(1.7)$ when using the function and when using Taylor's expansion with $a=0$ and 10 to 20 terms.

Compute probabilities and similar problems

- For each distribution R defines 4 functions:

  p*name*() distribution function
  q*name*() quantile function
  d*name*() density (probability) function
  r*name*() random numbers generation

  where *name* is the name of the distribution

  Example: pnorm() qnorm() dnorm() rnorm()

- To see which parameters are needed: use ?p*name*

- R standard installation defines many distributions (?distributions)

- For other distributions ..... google it
  Example: Tweedie distribution

Exercises 3 –

1. Assume that X is Poisson distributed with mean 7.2.
   a. Compute $\Pr(X = 2)$, $\Pr(X \le 10)$, $\Pr(2 \le X \le 10)$
   b. Find the smallest value of x such that $\Pr(X \le x) \ge 0.95$
2. Assume that X follows a normal distribution with mean 7.2 and standard deviation 2
   a. Compute $\Pr(X \le 7)$, $\Pr(6.5 \le X \le 9.5)$
   b. Find x such that $\Pr(|X - 7.2| \le x) = 0.95$
   c. Find the value of the density function at $x = 6$
3. (exercise 3 from exercises 1 sheet) Let $f(x)$ be the density function of a standardized normal variable. Compute $f(0)$ and $f(2)$.
4. Assume that X follows a gamma distribution with mean 7.2 and standard deviation 2. Compute $\Pr(X \le 7)$. Compute the median of the distribution.
5. Assume that you observed the following random sample from a normally distributed population 11.2 13.3 12.2 11.3 14.0 6.9 13.7 9.7 10.9 9.3. Compute a 95% confidence interval for μ and test $H_0 : \mu \le 10$ against $H_1 : \mu > 10$.

Matrices

- Creating matrices
  - From scratch → function matrix
    > mat=matrix(c(1,2,3,4,5,6),nrow=2,ncol=3,byrow=FALSE)
  - From vectors → functions cbind and rbind
    > x=1:7; y=x^2; mat_col=cbind(x,y) ; mat_row=rbind(x,y)
- Arithmetic operators operates element by element
- Some useful matrix functions
  - Transpose → t()
  - Matrix multiplication → %*% (remember that using * we get an element by element product - Hadamard product)
  - Inversion (to avoid due to numerical instability) → A.inv=solve(A)
  - Number of rows and of columns: nrow() and ncol()
  - rowSum(), rowMeans(), colSums() and colMeans()
  - Eigenvalues and eigenvectors → eigen()
  - Function diag(): can be used to extract the diagonal or to create a diagonal matrix (or an identity matrix)

- Subsetting with matrices: matrix $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$
  - Each element → A[2,3] gives 7
  - Each row → A[2,] gives (5 6 7 8)
  - Each column → A[,3] gives (3 7 11 15)
  - Extracting a sub-matrix → A[2:3,3:4] gives $\begin{bmatrix} 7 & 8 \\ 11 & 12 \end{bmatrix}$

- Linear system of equations. Let v be a column vector (9 12 15 18). To solve the linear system of equations $Ax = v$ we use solve(A,v)

Exercises 4

1. Define the matrix $M = \begin{bmatrix} 0.9 & 0 & 0.09 & 0 & 0.01 \\ 0.9 & 0 & 0 & 0.09 & 0.01 \\ 0 & 0.9 & 0 & 0 & 0.1 \\ 0 & 0 & 0.9 & 0 & 0.1 \\ 0 & 0 & 0 & 0.9 & 0.1 \end{bmatrix}$
2. Compute the transpose of matrix, i.e. $A = M^T$
3. Compute $M2 = M^2$ and compare the result with $B = M * M$
4. Compute $M10 = M^{10}$
5. Let v be the row vector $(0,0,1,0,0)$. Compute $y = vM$ and $z = Av^T$
6. Solve the linear system of equations $\begin{cases} 2x_1 + x_2 & = 5 \\ x_1 + x_3 & = 4 \\ x_1 + x_2 + x_3 & = 1 \end{cases}$
7. Replace the last row of matrix M by $(1,1,1,1,1)$ and call this new matrix D.
8. Compute matrix $I - M$

Using functions and data available in R

- To use functions and/or a data set available in one of R packages
  - Install that package (if necessary)
  - Load the package into R using the library() function. Alternatively use the menu (packages| load package
  - For a function, just use it
  - For a data set, extract it using the data() function
- How to choose the adequate package? *Google it*
  Example: "loss amount distributions modeling R" package actuar

Example: Assume that we want to use data set "ais" included in the package "alr3"

- Install "ais" package:
  - Choose a mirror
  - Download
- Load the package – use the menu utility or library(alr3) function
  Now all datasets and functions included in the package are available
- To get the dataset "ais" use data(ais)

Data on 102 male and 100 female athletes collected at the Australian Institute of Sport. This data frame contains the following columns:

Sex	(0 = male or 1 = female)
Ht	height (cm)
Wt	weight (kg)
LBM	lean body mass
RCC	red cell count
WCC	white cell count
Hc	Hematocrit
Hg	Hemoglobin
Ferr	plasma ferritin concentration
BMI	body mass index, weight/(height)**2
SSF	sum of skin folds
Bfat	Percent body fat
Label	Case Labels
Sport	Sport

Source
Ross Cunningham and Richard Telford

Reading datasets from other sources

- **read.table()** more versatility
- read.csv()
- scan()
- package “foreign”

Example: read ceo_salaries file

```
data1=read.csv("C:/.../CEO_salary_Wooldridge.csv",sep=";")
```

or

```
data2=read.table("C:/.../CEO_salary_Wooldridge.csv",header=TRUE,sep=";")
summary(data1)
```

Data frame contents

1. salary	1990 compensation, $1000s
2. age	in years
3. comten	years with company
4. ceoten	years as ceo with company
5. sales	1990 firm sales, millions
6. profits	1990 profits, millions
7. mktval	market value, end 1990, mills.

Source: Wooldridge, W. (2013) Introductory Econometrics, 5th ed., Thomson

Working with data frames

Data frame is a more general structure than a matrix in the sense that different columns may have different modes (numbers, strings, logical, …)

Each column has a label and each row also has a label

- Naming the variables: dataframe$variable
- Attach() and detach() functions
- Names()

Example

```
summary(data1)
data1
salary
data1$salary
attach(data1)
salary
detach(data1)
salary
head(data1)
```

Linear regression using data frames

- function lm()
- Examples: Can we explain relate CEO salaries with some characteristics of their companies?
  lm(salary~sales+mktval+comten+ceoten,data=data1)

Loops and conditional execution

Looping in R is **slow.** Speed mostly depends on what is occurring inside the loop(s) and the size of the data. Vectorization is usually faster than looping.

Conditional execution:

- Command if: if (*condition*) command1 [else command2]
  - Command1 can be a command or a block of commands inside {} brackets
  - If applied to vectors only the first element is tested (to use very carefully)
  - The last part of the command (inside [] brackets) is optional
  - Example 1: Assume that p.value is the p-value of a test that is to be performed using a 5% significance level.
    if (p.value>=0.05) print("Do not reject H0") else print("reject H0")
  - Example 2: x=1:5; if (x>=3) print("TRUE") else print("FALSE")

- Command ifelse: ifelse(*condition*,command1,command2)
  - If condition is TRUE command1 is executed otherwise command2
  - If applied to vectors each element is tested
  - Example: x=1:5; ifelse (x>=3,print("TRUE"),print("FALSE"))

Looping:

- Command for: for (*condition*) command1
  - *Condition* is something like x in 1:10 or x in a where a is a vector;
  - Command1 is usually a block of commands inside {} brackets
  - 2 special commands **can** be used:
    - next Moves the iterator to the next element and continues at the start of the loop
    - break Immediately exits the loop
  - Example 1: for(i in 1:10) print(i)
  - Example 2: print only the odd numbers and exits when x=7 without printing x

```
for(x in 1:9){
  if (x%%2==0) next        # if x is even go to the next value without printing
  if (x==7) break          # if x=7 exits the loop
  print(x)
  }
```

  Or a=seq(1,5,by=2); for(x in a) print(x)

- Commands while and repeat – To avoid

- Command while: while (*condition*) command1
  - Command1 is repeatedly executed until condition becomes FALSE
  - Command1 is usually a block of commands inside {} brackets
  - The special commands next and break **can** be used
  - The ESC key
  - Example 2: x=1; while(x<=5){ print(x); x=x+2}
- Command repeat: repeat command1
  - Command1 is usually a block of commands inside {} brackets
  - Command1 is repeatedly executed until a break is founded. Consequently break is mandatory to avoid an endless loop
  - The special command next **can** be used
  - The ESC key
  - Example 2: x=1; repeat{ print(x); x=x+2; if(x>5) break}

Exercises 5 -

1. Find an approximate solution for the equation $2x\ln(x)-3=0$ with an error less than 1e-06
   a. Using command repeat
   b. Using command while

Solution

Let $P(x)=2x\ln(x)-3$. As we know that $P(1)=-3<0$, $P(3)=6\ln(3)-3\approx 6.59>0$ and that $P(x)$ is a continuous function in this interval we can iterate until we get the solution.

We can also use function uniroot (among different options).

Plotting

Graphical facilities are an important component of the R environment. We will only discuss the very basic plotting functions

Plotting commands are divided into three basic groups:

- High-level plotting functions create a new plot on the graphics device, possibly with axes, labels, titles and so on.
- Low-level plotting functions add more information to an existing plot, such as extra points, lines and labels.
- Interactive graphics functions allow you interactively add information to, or extract information from, an existing plot, using a pointing device such as a mouse. *Out of the scope of this introduction*.

In addition, R maintains a list of graphical parameters which can be manipulated to customize your plots.

High-level plotting commands

High-level plotting commands always **start a new plot**, erasing the current plot if necessary **unless the option** add=TRUE is specified.

- The plot() function: the type of plot produced is dependent on the type or class of the first argument (see help function for more details). The more usual situations are
  - plot(x, y) - If x and y are vectors, plot(x, y) produces a scatterplot of y against x.
  - plot(*function*,xlower,xupper) – plot a function. Example: plot(cos, -pi, 3*pi)
- Other high-level graphics functions produce different types of plots. Some examples are:
  - QQplots as qqnorm(x) or qqplot(x, y) - The first form plots the numeric vector x against the expected Normal order scores (a normal scores plot) and the second form plots the quantiles of x against those of y to compare their respective distributions.
  - Histograms: Produces a histogram of the numeric vector x. Commands hist(x) or hist(x, nclass=n) or hist(x, breaks=b, ...). If the probability=TRUE argument is given, the bars represent relative frequencies divided by bin width instead of counts.

Many options are available using the adequate arguments which may be passed to high-level graphics functions. Some examples:

- add=TRUE superimpose the plot on the current plot (some functions only).
- axes=FALSE suppresses generation of axes — useful for adding your own custom axes with the axis() function. The default is axes=TRUE.
- type= controls the type of plot produced, as follows:
  - type="p" Plot individual points (the default)
  - type="l" Plot lines
  - type="b" Plot points connected by lines (both)
  - type="o" Plot points overlaid by lines
  - type="h" Plot vertical lines from points to the zero axis (high-density)
  - ...
- xlab=*string* and or ylab=*string* - Axis labels for the x and y axes.
- main=*string* Figure title, placed at the top of the plot in a large font.
- sub=*string* Sub-title, placed just below the x-axis in a smaller font.

Low level plotting commands

Some of the more useful low-level plotting functions are:

- points(x, y) or lines(x, y) - Adds points or connected lines to the current plot. plot()'s type= argument can also be passed to these functions (and defaults to "p" for points() and "l" for lines().)
- abline(a, b) Adds a line of slope b and intercept a to the current plot
- abline(h=y) Adds an horizontal line to the current plot
- abline(v=x) Adds a vertical line to the current plot
- ...

Exercises 6 – Use data set CEO_salary_Wooldridge.csv

1. Draw a histogram for the salary variable. Add a vertical line in your histogram at the mean value of the variable
2. Draw a scatter plot using of salary against sales. Add a regression line in your scatter plot.

User-written functions:

- 2 steps procedure: First define (write) the function and, then, use it
- How to define a function:

  name=function(*arguments*){

  commands (use return() to send back an answer)

  }
- To use the function, proceed as usual

Example

```
My.sum=function(x,y){ return(x+y)}
x=2:5; y=c(1,3,7);
my.sum(x,y)  or my.sum(2:5,c(1,3,7))  or z= my.sum(x,y)
```

If the function returns a data structure that is more complex use a list structure
Variables defined inside the function are local. They cannot be accessed in the main program.
You can give a default value for each argument

Exercises 7:

1. Write a function such that if the argument is $v=(x_1,x_2,\cdots,x_n)$ then the function returns the vector of moving averages $\left(\frac{x_1+x_2+x_3}{3},\frac{x_2+x_3+x_4}{3},\cdots,\frac{x_{n-2}+x_{n-1}+x_n}{3}\right)$
   Check your answer using the vector c(1:5,6:1)
2. Consider the continuous function
$$f(x)=\begin{cases}x^2+2x+3 & x<0\\ x+3 & 0\le x<2\\ x^2+4x-7 & 2\le x\end{cases}$$
   Write a function that uses x (or a vector of values for x) and return the corresponding value(s) for $f(x)$
3. Write a function that uses and observed sample (assume that the population is normally distributed) and returns the confidence intervals. The level of confidence should be an argument for this function.
4. Write a function that receives a positive integer as arguments and returns TRUE if the number is prime.

Writing results to a file

- Use cut and paste
- Use write.table()
- Use adequate packages

Example: Define matrix $M=\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix}$ and write this matrix to a file that can be easily imported to EXCEL

```
> M=matrix(1:15,nrow=3,byrow=TRUE)
> getwd()       #if you need to check which is the working directory
> write.table(M,"Rcourse.txt",row.names=F,col.names=F,sep=";")
```

You can try different alternatives and see what happens

```
> write.table(M,"cursoR.txt")
> write.table(M,"cursoR.txt",append=TRUE,row.names=F,col.names=F)
```