

A briefly Introduction to R language

M. Beccuti

Università degli Studi di Torino

November 2019



The R Project

- Environment for statistical computing and graphics;
- Free software and Open-source;
- A simple programming language:
 - ▶ it is an open-source implementation of S language;
 - ▶ it is among the Top 10 Programming Languages in 2018 for *IEEE Spectrum Journal*;



- software and packages can be downloaded from:

www.cran.r-project.org

- Versions of R exist of Windows, MacOS, Linux and various other Unix-like OS.

Why to use R language

- Implement many common statistical and bioinformatics procedures;
- Provide excellent graphics functionality;
- A convenient starting point for many data analysis projects
- Libraries (namely packages) can be automatically downloaded from:

www.cran.r-project.org
<https://www.bioconductor.org/>

- It is standard for data mining and statistical analysis;
- Efficient data structures make programming easier.



Download and Install R language

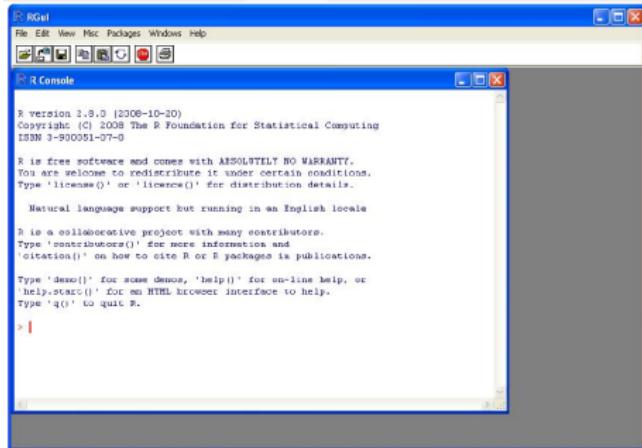
The screenshot shows a web browser window with the URL cran.r-project.org in the address bar. The page title is "The Comprehensive R Archive Network". On the left, there's a sidebar with links for CRAN Mirrors, What's new?, Task Views, Search, About R, R Homepage, The R Journal, Software, R Sources, R Binaries, Packages, Other, Documentation, Manuals, FAQs, and Contributed. The main content area has a heading "Download and Install R" and a sub-section "Precompiled binary distributions of the base system and contributed packages, Windows and Mac users most likely want one of these versions of R:". It lists three download links: "Download R for Linux", "Download R for (Mac) OS X", and "Download R for Windows". Below this, it says "R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above." Another section, "Source Code for all Platforms", explains that Windows and Mac users should download precompiled binaries, while Linux users should use their package manager. It lists several bullet points about source code availability:

- The latest release (2013-09-25, Frisbee Sailing) [R-3.0.2.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).

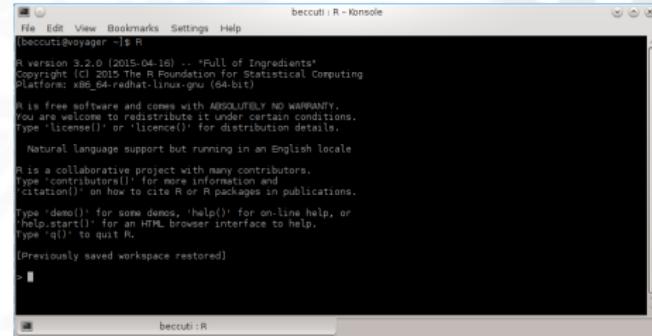
<http://cran.mirror.garr.it/mirrors/CRAN/>

Download the appropriate version (w.r.t. your OS) and follow the instructions to install the program.

R under GUI



from Windows



from Linux

R under GUI using Rstudio

RStudio allows the user to run R in a more user friendly environment.

It is free-software and available at <http://www.rstudio.com/>

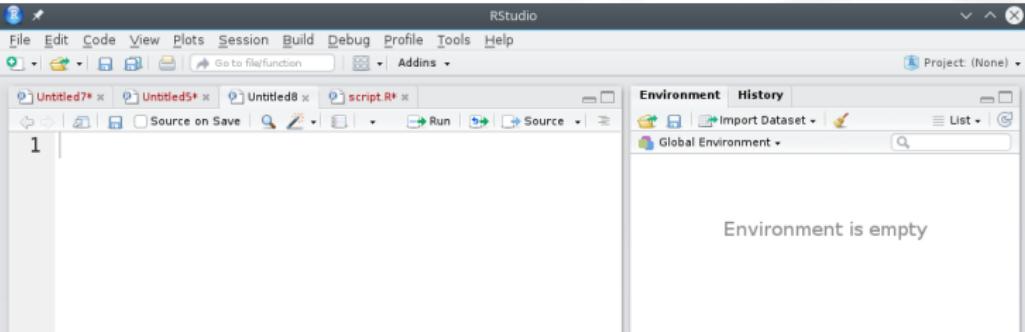
The screenshot shows the RStudio interface with the following sections:

- Console:** Displays the R command-line interface, including the R version information, license terms, and a sample matrix creation command.
- Environment:** Shows the current workspace objects, including a variable 'm' which is a 1x20 integer vector from 1 to 20.
- Files:** Lists files in the workspace, including 'ClientServer.eps' and 'ProbNetOnlySecurity.eps'.
- Help:** Provides help for various R functions and packages.

Annotations in green text provide additional information:

- Console is where you can type commands and see output** (points to the Console tab)
- 1. Workspace tab shows all the active objects**
- 2. History tab shows a list of commands recently used** (points to the History tab in the Environment panel)
- 1. Files tab shows all the files and folders in your workspace.**
- 2. Plots tab will show all your graphs.**
- 3. Packages tab will list a series of packages or add**
- 4. Help tab can be used for additional info** (points to the Help tab in the bottom right)

RStudio prompt and script



The screenshot shows the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The Project dropdown is set to (None). The left sidebar has tabs for Untitled7*, Untitled5*, Untitled8*, and script.R*. The main workspace shows a single digit '1' at the top. The Environment pane on the right displays the message "Environment is empty". A blue arrow points from the text "R Script" in the bottom-left corner towards the Environment pane. The bottom-left pane is labeled "Console" and contains the following text:

```
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.  
  
[Workspace loaded from ~/.RData]  
>
```

The bottom-right pane is titled "Solver for Ordinary Differential Equations (ODE)" and provides documentation for the lsode function.

Solver for Ordinary Differential Equations (ODE)

Description

Solves the initial value problem for stiff or nonstiff systems of ordinary differential equations (ODE) in the form:

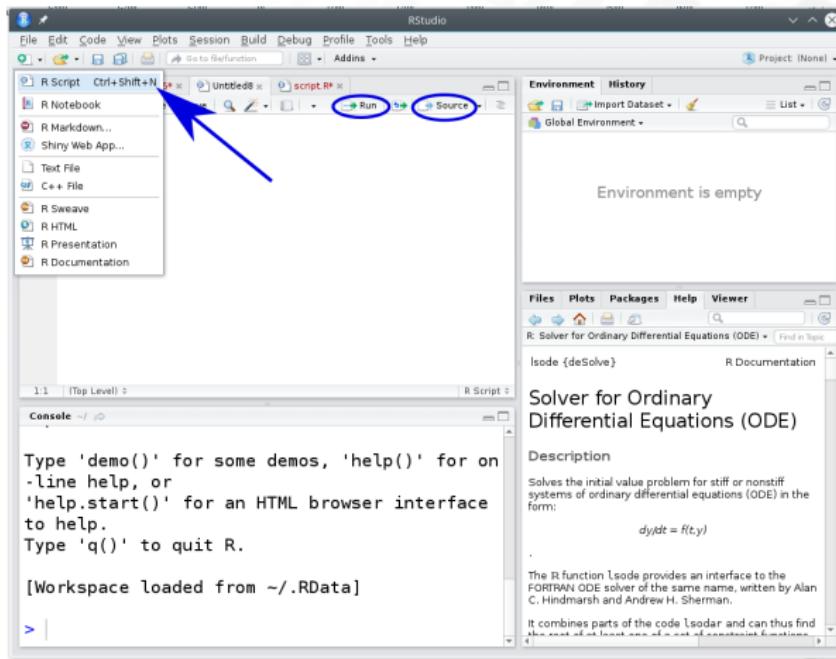
$$\frac{dy}{dt} = f(t,y)$$

The R function lsode provides an interface to the FORTRAN ODE solver of the same name, written by Alan C. Hindmarsh and Andrew H. Sherman.

It combines parts of the code lsodar and can thus find the root of at least one of a set of constraint functions.

RStudio prompt and script

- R script can be used to save R commands into a file;
- Commands into R script can be executed line by line (clicking on Run) or globally (clicking on Source).



RStudio prompt and script

- Commands can be directly typed into the R script console.

The screenshot shows the RStudio IDE. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The main window has tabs for 'script.R' and 'Untitled8.R'. The left pane contains a code editor with the following content:

```
1 4*2
2 |
```

A blue arrow points from the text '4*2' in the editor to the R Script tab in the bottom-left corner. The bottom-left panel is the 'Console' tab, which displays:2.1 (Top Level) R Script
Console ->
line help, or
'help.start()' for an HTML browser interface
to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> 4*2
[1] 8
> |

The right side of the interface shows the 'Environment' tab with the message 'Environment is empty'. Below it is the 'Files' tab. A large window on the right is titled 'R: Solver for Ordinary Differential Equations (ODE)' and shows the documentation for the 'lsode' function, which is part of the 'deSolve' package. The page title is 'Solver for Ordinary Differential Equations (ODE)' and the section title is 'Description'. It describes the function as solving initial value problems for stiff or nonstiff systems of ordinary differential equations (ODE) in the form $dy/dt = f(t,y)$. The text also mentions that the R function 'lsode' provides an interface to the FORTRAN ODE solver of the same name, written by Alan C. Hindmarsh and Andrew H. Sherman.

Assignments in R

It is often required to store intermediate results so that they do not need to be re-typed over and over again. To assign a value of 324 to the variable X type:

```
> X <- 324
```

or

```
> X = 324
```

Variable X can be used in next expressions:

Example

```
> X  
[1]324
```

```
> X = X + X; X  
[1]648
```

```
> X + X  
[1]648
```

```
> X/4  
[1]162
```

```
> sqrt(X)  
[1]18
```

```
> X^sqrt(X)  
[1]1.54814e+45
```

Variable name in R

R is a case-sensitive language, hence x and X do not refer to the same variable.

Variable name:

- can be created using letters, digits and the . (dot) symbol;

```
> data1.address  
> d14.f
```

- must not start with **a digit** or a . followed by a digit.
- some names are reserved by the system: *if, while, NULL, TRUE ...*

Variable type in R

Basic variable types are:

Numeric: integer, floating point values;

Boolean: values corresponding to **True** or **False**;

Strings: sequences of characters.

Type is determined automatically when variable is created with `<-` or `=` operator.

Data structures/Objects are:

R provides types of different object.

Vector: a collection of elements (numbers, logical values and character strings) with same type;

Array: a generalization of a vector;

List: collections of objects of any type;

e.g. list of vectors, list of matrices, etc.

Data Frame an array in which the type of each element can be different;

Factor takes on a limited number of values;

Getting help in R

R provides a built-in help facility.

- To get more information on any specific function, e.g. `sqrt()`, the command is:

`help(sqrt)`

or

`?sqrt`

- help on features specified by special characters must enclose in single or double quotes (e.g. `"[]"` `help("[]")`)
- Help is also available in HTML format by running `help.start()`
- For more information use
`?help`

Argument Matching in R

How does R know to match arguments?

Argument matching is done in a few different ways:

- The arguments are matched by their positions. The first supplied argument is matched to the first formal argument and so on.

`> myfun(3, 4, 7) x=3, y=4 and p=7`

- The arguments are matched by name. A named argument is matched to the formal argument with the same name:

`> myfun(y = 4, x = 3, p = 7) x=3, y=4 and p=7`

- Name matching happens first, then positional matching is used for any unmatched arguments.

Getting help in Rstudio

The screenshot shows the RStudio interface with three main panes:

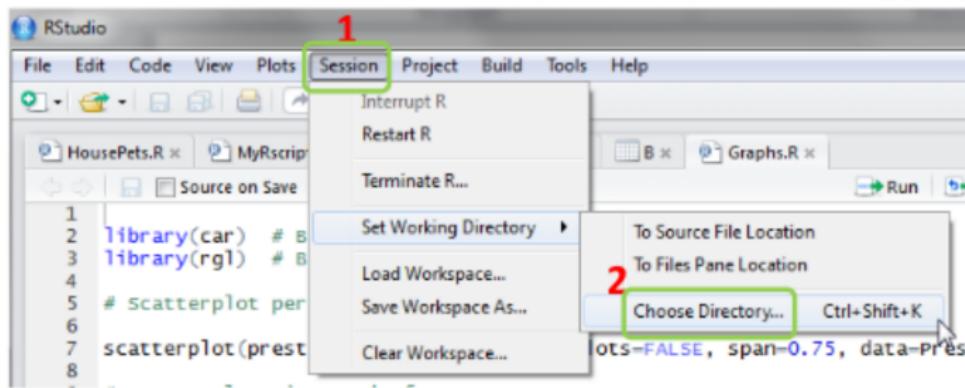
- Console** (left): Displays the R startup message, license information, natural language support, collaborative project details, and a command history block starting with `> m = matrix(1:20,ncol=5)`.
- Environment** (top right): Shows the global environment with a variable `m` of type `int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ...`.
- Help Viewer** (bottom right): Displays the documentation for the `ls` function. The title is "List Objects". The "Description" section states: "ls and objects return a vector of character strings giving the names of the objects in the specified environment. When invoked with no argument at the top level prompt, ls shows what data sets and functions a user has defined. When invoked with no argument inside a function, ls returns the names of the function's local variables: this is useful in conjunction with browser." The "Usage" section shows the function signature: `ls(name, pos = -1L, envir = as.environment(pos), all.names = FALSE, pattern, sorted = TRUE)` and `objects(name, pos = -1L, envir = as.environment(pos), all.names = FALSE, pattern, sorted = TRUE)`.

Working directory

Working directory in R:

- Working directory contains data and R scripts. It is a directory of the file-system;
- `getwd()` returns the current Working directory;
- `setwd("new_path")` sets Working directory;

Working directory in RStudio:



Packages in R

- R provides libraries of packages. Packages contain various functions and data sets for numerous purposes;
- Some packages are part of the basic installation. Others can be downloaded from CRAN:
`> install.packages("devtools")`
- To use functions and data sets of a package, it must be loaded into the workspace:
`> library(devtools)`
- To check what packages are currently loaded into the workspace:
`> search()`
- A loaded package can be removed:
`> detach("package:devtools")`

Observe:

if you terminated your session and start a new session with the saved workspace, you must load the packages again.

Vector in R

Vectors in R

- an ordered list of homogeneous elements;
- Vectors are the simplest type of object in R;
There are 3 main types of vectors:
 - ▶ Numeric vectors;
 - ▶ Character vectors;
 - ▶ Logical vectors.

- To create a numeric vector x consisting of 6 numbers, 1.4, 6, 23.1, 65.43, 2.7, 55 use:

```
> x = c(1.4, 6, 23.1, 65.43, 2.7, 55)
```

or

```
> assign("x", c(1.4, 6, 23.1, 65.43, 2.7, 55))
```

Numeric vectors in R

- To print the contents of x:

```
> x  
[1] 1.4 6 23.1 65.43 2.7 55
```

symbol [1] in front of the result is the index of the first element in the vector x.

- To access a particular element of x:

```
> x[1]  
[1] 1.4
```

```
> x[6]  
[1] 55
```

```
> x[c(1, 6)]  
[1] 1.4 55
```

```
> x[-c(1, 5)]    Operator - means: select all the elements except those ....  
[1] 6 23.1 65.43 55
```

Numeric vectors in R

- To modify a particular vector element:

```
> x[2] = 5  to modify the 2nd element of x in 5  
[1]1.4 5 23.1 65.43 2.7 55
```

```
> x[4] = 5  
[1]1.4 5 23.1 5 2.7 55
```

- To modify more than one vector elements:

```
> x[c(2,4)] = c(6,65.43)  
[1]1.4 6 23.1 65.43 2.7 55
```

```
> y = x  
> y[y < 3] = 1  
> y  
[1]1 6 23.1 65.43 1 55
```

Numeric vectors in R

- A vector can be used to do further assignments:

```
> y = c(x, 2, 3, x[c(1, 3)])
```

vector y with 10 entries is created:

```
> y  
[1] 1.4 6 23.1 65.43 2.7 55 2 3 1.4 23.1
```

- Operations are performed on each single element:

```
> x/10  
[1] 0.14 0.6 2.31 6.543 0.27 5.5
```

- Short vectors are “recycled” to match long ones (if it is possible):

```
> v = x[c(1, 2)] + y    x[c(1, 2)] is repeated 5 times  
> v  
[1] 2.8 12 24.5 71.43 4.1 61 3.4 9 2.829.1
```

How to generate sequences in R

- In R it is possible to generate sequences of numbers

- ▶ using operator “`:`”

```
> 1 : 5
```

```
[1] 1 2 3 4 5
```

- ▶ using function `seq()`

```
> seq(1, 5)
```

```
[1] 1 2 3 4 5
```

```
> seq(from = 1, to = 5)
```

```
[1] 1 2 3 4 5
```

We can also specify a step size (using `by=value`) or a length (using `length.out=value`) for the sequence.

```
> seq(1, 5, by = 0.5)
```

```
[1] 1 1.5 2 2.5 3 3.5 4 4.5 5
```

```
> seq(from = 1, to = 5, length.out[= 9])
```

```
[1] 1 1.5 2 2.5 3 3.5 4 4.5 5
```

- ▶ using function `rep()`

```
> rep(x, 3)
```

```
[1] 1.40 6.00 23.10 65.43 2.70 55.00 1.40 6.00 23.10 65.43 2.70 55.00
```

```
[13] 1.40 6.00 23.10 65.43 2.70 55.00
```

Character vector in R

- A string is identify by " "
- A string vector is defined as well as number vector by `c()` operator
`> y = c("ROMA", "MILANO", "TORINO")`
- several functions in R to manipulate character vectors.
`paste, as.character, is.character, strsplit, substr...`

```
> paste("HOME", "WHILE", "DOG", sep = " : ")  
[1]"HOME : WHILE : DOG"  Concatenate char vectors
```

```
> x = c(1, 3, 45, 7)  
> is.character(x)  test if an object is of type character  
[1]FALSE
```

```
> is.character(as.character(x))  
[1]TRUE
```

Data Frame in R

Data Frame in R

- It is used to storage data table in R;
- It can be considered as a matrix in which columns can contain different types;
- We can create data frames from pre-existing variables:

```
> name = c("GENE1", "GENE2", "GENE3")
> seq = c("ATCCT..", "CCTTT..", "CCAACT..")
> count = c(100, 20, 4)
> d = data.frame(name, seq, count)
> d
```

	<i>name</i>	<i>seq</i>	<i>count</i>
1	<i>GENE1</i>	<i>ATCCT..</i>	100
2	<i>GENE2</i>	<i>CCTTT..</i>	20
3	<i>GENE3</i>	<i>CCACT..</i>	4

Data Frame in R

Main operations:

- `attributes(d)` returns the data frame attributes:

```
> attributes(d)
```

```
$names
```

```
[1]"name" "seq" "count"
```

```
$row.names
```

```
[1]1 2 3
```

```
$class [1]"data.frame"
```

- `colnames(d)` returns the names of data frame columns:

```
> colnames(d)
```

```
[1]"name" "seq" "count"
```

```
> colnames(d) = c("c1", "c2", "c3", "c4")    change column names.
```

- `rownames(d)` returns the names of data frame rows:

```
> rownames(d)
```

```
[1]1 2 3
```

Indexing Data Frame in R

- it is possible to use the same method of matrices to access values of a data frame.

```
> d
```

	<i>name</i>	<i>seq</i>	<i>count</i>
1	<i>GENE1</i>	<i>ATCCT..</i>	100
2	<i>GENE2</i>	<i>CCTTT..</i>	20
3	<i>GENE3</i>	<i>CCACT..</i>	4

`> d[2, 2]` gives the value in the 2nd row and 2nd column of *d*.
`[1] CCTTT..`

`> d[2,]` gives the values in the 2nd row of *d*.
`[1] GENE2 CCTTT.. 20`

`> d[, 3]` gives the values in the 3rd column of *d*.
`[1] 100 20 4`

Data Frame in R

Main operations(2):

- `summary(d)` returns a summary of data frame:

```
> summary(d)
```

<i>name</i>	<i>seq</i>		<i>count</i>
<i>GENE1</i> : 1	<i>ATCCT..</i> : 1	<i>Min. :</i>	4.000
<i>GENE2</i> : 1	<i>CCTTT..</i> : 1	<i>1stQu. :</i>	12.00
<i>GENE3</i> : 1	<i>CCACT..</i> : 1	<i>Median :</i>	20.00
		<i>Mean :</i>	41.33
		<i>3rdQu. :</i>	60.00
		<i>Max. :</i>	100.00

- `subset(d,cond)` returns a subset of rows according to condition:

```
> subset(d, d[,3] > 10)
```

	<i>name</i>	<i>seq</i>	<i>count</i>
1	<i>GENE1</i>	<i>ATCCT..</i>	100
2	<i>GENE2</i>	<i>CCTTT..</i>	20

I/O in R language

Input from a file in R

- R provides a set of functions to read data from files:
 - ▶ `read.table()` is used to read data frames from formatted text files.
A variable separator can be specified.
 - ▶ `read.csv()` is used to read data frames from comma separated variable files.
 - ▶ `read.csv2()` is used to read data frames from semicolon separated variable files.
 - ▶ `load()` is used to reload datasets written with the function `save()`.
Data are stored in binary format (more compact!!).

Input from a file in R

- `read.table()` reads a file in table format and creates a data frame from it,

```
read.table(file, header=FALSE, sep = " ", dec=".",
           stringAsFactors=TRUE ...)
```

`file` : the name of the file in which the data are stored;

`header` : a logical value indicating whether the file contains the names of the variables as its first line;

`sep` : the field separator character;

`dec` : the character used for decimal points;

`stringAsFactors` : logical: should character vectors be converted to factors?;

`row.names` : it can be a vector giving the actual row names, or a single number giving the column of the table which contains the row name;

`...` : optional arguments;

```
> d = read.table("./example.txt", header = TRUE, sep = "!")
```

```
> b = read.table("./example1.txt", header = FALSE, sep = " ")
```

Input from a file in R

- `read.csv()` reads a file in table format and creates a data frame from it,

```
read.csv(file,header=FALSE, sep=",", dec=".")
```

`file` : the name of the file in which the data are stored;

`header` : a logical value indicating whether the file contains the names of the variables as its first line;

`sep` : the field separator character;

`dec` : the character used for decimal points;

`stringAsFactors` : logical: should character vectors be converted to factors?;

`row.names` : it can be a vector giving the actual row names, or a single number giving the column of the table which contains the row name

`...` : optional arguments;

```
> d = read.csv("./example.txt", header = TRUE)
```

```
> b = read.csv("./example1.txt", header = FALSE)
```

Input from a file in R

- `read.csv2()` reads a file in table format and creates a data frame from it,

```
read.csv2(file,header=FALSE, sep=";", dec=".")
```

`file` : the name of the file in which the data are stored;

`header` : a logical value indicating whether the file contains the names of the variables as its first line;

`sep` : the field separator character;

`dec` : the character used for decimal points;

`stringAsFactors` : logical: should character vectors be converted to factors?;

`row.names` : it can be a vector giving the actual row names, or a single number giving the column of the table which contains the row name

`...` : optional arguments;

```
> d = read.csv2("./example.txt", header = T)
```

```
> b = read.csv2("./example1.txt", header = F)
```

Input from a file in R

- `load()` reload datasets written with the function `save()`.

```
load(file, ...)
```

File : the name of the file in which the data are stored;
`verbose = FALSE` : if TRUE item names are printed;
... : optional arguments;

```
> load("./example.data")
```

```
> load("./example.data", verbose = T)
```

Loading objects :

m

Writing a file in R

- R provides a set of functions to write data into files:
 - ▶ `write.table()` is used to write data frames into formatted text files.
A variable separator can be specified.
 - ▶ `write.csv()` is used to write data frames into comma separated variable files.
 - ▶ `write.csv2()` is used to write data frames into semicolon separated variable files.
 - ▶ `save()` is used to save datasets into a binary file.
Data are stored in binary format (more compact!!).

Writing a file in R

- `write.table()` is used to write data frames into formatted text files ,

```
write.table(x,file,col.names=TRUE,row.names=TRUE, sep=" ", dec=". ", ...)
```

`x` : the object to be written;

`file` : the name of the file in which the data are stored;

`col.names` : if TRUE column names are stored;

`row.names` : if TRUE row names are stored;;

`sep` : the field separator character;

`dec` : the character used for decimal points;

`...` : optional arguments;

```
> write.table(b, "./example.txt", col.names = TRUE, row.names =  
TRUE, sep = "!")
```

```
> write.table(b, "./example.txt", col.names = FALSE, row.names =  
FALSE, sep = ", ")
```

Writing a file in R

- `write.csv()` is used to write data frames into formatted text files ,

```
write.csv(x,file,col.names=TRUE,row.names=TRUE, sep=",", dec=".")
```

`x` : the object to be written;

`file` : the name of the file in which the data are stored;

`col.names` : if TRUE column names are stored;

`row.names` : if TRUE row names are stored;;

`sep` : the field separator character;

`dec` : the character used for decimal points;

`...` : optional arguments;

```
> write.csv(b, “./example.txt”, col.names = TRUE, row.names = TRUE)
```

```
> write.csv(b, “./example.txt”, col.names = FALSE, row.names = FALSE)
```

Writing a file in R

- `write.csv2()` is used to write data frames into formatted text files ,

```
write.csv2(x,file,col.names=TRUE,row.names=TRUE, sep=";",dec=",", ...)
```

`x` : the object to be written;

`file` : the name of the file in which the data are stored;

`col.names` : if TRUE column names are stored;

`row.names` : if TRUE row names are stored;;

`sep` : the field separator character;

`dec` : the character used for decimal points;

`...` : optional arguments;

```
> write.csv2(b, “./example.txt”, col.names = TRUE, row.names = TRUE)
```

```
> write.csv2(b, “./example.txt”, col.names = FALSE, row.names = FALSE)
```

Writing a file in R

- `save()` writes an external representation of R objects to the specified file,

```
save(...,file, ...)
```

... : a list of objects to be saved;

file : the name of the file in which the data are stored;

... : optional arguments;

```
> save(b, c, file = "./example.data")
```