Introduction to R

R Tutorial - v1.0.1

Jean-Yves Sgro
This section is based on Emmanuel Paradis’s “R for beginners” which can be downloaded from:

http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf
(English, 72 pages) or:

http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf
(French, 77 pages). Or

http://cran.r-project.org/doc/contrib/rdebuts_es.pdf
(Spanish, 60 pages, translated by Jorge A. Ahumada, 2003)

Therefore the following Copyright notice applies:


Permission is granted to make and distribute copies, either in part or in full and in any language, of this document on any support provided the above copyright notice is included in all copies. Permission is granted to translate this document, either in part or in full, in any language provided the above copyright notice is included.

Additional material is © Jean-Yves Sgro (2007-2013) and subject to permissions identical to those above.

Within the text: user input is shown as **bold text or commands**

As much as possible, R commands and R output screen text are shown written with single space fonts such as `courier, monaco`
# Table of Contents

Foreword ................................................................................................................................. 3

R ............................................................................................................................................. 5

R Concepts ............................................................................................................................... 5

How R works ........................................................................................................................... 6

Intro. & Preparations ............................................................................................................... 9

Starting R ................................................................................................................................ 10

R objects .................................................................................................................................. 11
  1. Simplest, implicit command .............................................................................................. 11
  2. The “assign” operator (= or <-) : create, list and delete object in memory ........... 12
  3. Online help ......................................................................................................................... 14

Data with R ............................................................................................................................ 16
  1. R Objects .......................................................................................................................... 16
  2. Reading data from a file .................................................................................................... 18
  3. Saving data into a file ........................................................................................................ 20
  4. Generating data ............................................................................................................... 21
     4.1. Regular sequences ........................................................................................................ 21
     4.2. Random sequences ...................................................................................................... 24
  5. Manipulating objects ......................................................................................................... 27

Graphics with R ...................................................................................................................... 31
  1. Plotting symbols ............................................................................................................... 33
  2. Split screen multiple plots ............................................................................................... 35

Appendix A: R outside SBGrid .............................................................................................. 39

The Comprehensive R Archive Network ................................................................................. 39
Foreword

This tutorial was originally developed by JYS based on E. Paradis’s “R for beginners” manual for the purpose of a week-long course on microarray data analysis.

It is adapted here for use with R using the version installed with SBGrid.

However, a local installation can be used as well since all commands shown are standard R commands.

To install a local copy of R find the download link on the R Project web page www.r-project.org appropriate to your computing platform.

It should be noted that R is updated every 6 months. While the commands shown here are rather standard, basic commands, there can be differences arising as time passes.
The R language allows the user, for instance, to program loops to successively analyze several data sets. It is also possible to combine, in a single program, different statistical functions to perform more complex analyses.

At first, R could seem too complex for a non-specialist. This may not be true actually. In fact, a prominent feature of R is its flexibility. Whereas a classical software displays immediately the results of an analysis, R stores these results in an “object”, so that an analysis can be done with no result displayed.

R Concepts

Once R is installed on your computer, the software is executed by launching the corresponding executable. The prompt “>” indicates that R is waiting for your command.

Some specific of the commands can be executed with pull-down menu or icons (Mac and Windows).

At this stage, a new user is likely to wonder “What do I do now?” It is indeed very useful to have a few ideas on how R works when it is used for the first time, and this is what we will see now.

We shall see first briefly how R works. Then, I will describe the “assign” operator that allows creating objects, how to manage objects in memory, and finally how to use the on-line help which is very useful when running R.
How R works

When R is running, variables, data, functions, results, etc., are stored in the active memory (RAM) of the computer in the form of objects that have a name. The user can perform actions on these objects with operators (arithmetic, logical, comparison, . . .) and functions (which are themselves objects). The use of operators is relatively intuitive. We will see the details later. An R function may be sketched as follows:

All the actions of R are done on objects stored in the active memory of the computer (RAM): no temporary files are used (Figure 1).

The readings and writings of files are used for input and output of data and results (text tables, graphics, . . .). The user executes the functions with commands. The results are displayed directly on the screen, stored in an object, or written on the disk (particularly for graphics). Since the results are objects as well, they can be considered as data and further analysed as such. Data files can be read from the local disk or from a remote server through Internet.

Figure 1: A schematic view of how R works

---

1 Special packages exist for extreme cases that deal with unusually large data on computers with limited RAM. For examples packages ff or aroma.
R functions are all stored in packages within a library localized on the user’s hard drive called R_HOME/library (where R_HOME is the directory where R is installed.

On Windows, typically C:\Program Files\R\R-3.0.1;

on Macintosh: e.g.
/Library/Frameworks/R.framework/Versions/3.0/Resources/library/)

This directory contains packages of functions, which are themselves structured in directories. The package base is in a way the core of R and contains the basic functions of the language, particularly, for reading and manipulating data.

Each package has a directory called R with a file named like the package (for instance, for the package base, this is the file
R_HOME/library/base/R/base).

This file contains all the functions of the package.

On SBGrid the Framework is located on the “Groups” mounted volume (necessary for SBGrid; see SBGrid installation document.)
SBGrid is only available for Mac/Linux.

If you want to use R on Windows you have to install it first as a local installation from the R Project web site [www.r-project.org](http://www.r-project.org)

To use R on SBGrid first activate SBGrid on your system by double-clicking on the SBGrid logo (see SBGrid@Biochem document.)

One SBGrid has been activated, a Terminal with all SBGrid functions will be available. To check which version(s) of R is available type:

```
$ sbgrid -l R
```

Version information for: /programs/i386-mac/r

Default version: 3.2.1
In-use version: 3.2.1
Other available versions: 3.2.2 3.0.3 2.13.0 2.11.1
Overrides use this shell variable: R_M

Replace -l with -L to know versions for all platforms (change lower case “el” to uppercase “el” letter; this is not the number one.)

Check the environment variable associated with the default version:

```
$ printenv R_M
```

3.0.3

If for any reason you need to use an older version of R check the override method on the web site: [sbgrid.org/wiki/usage/versions](http://sbgrid.org/wiki/usage/versions)

However, the older versions currently offered on the list do not work.

It should also be noted that the SBGrid version might not be the most current version available for a local installation.
Starting R

On a newly activated SBGrid terminal simply type the letter R at the prompt to run the program. The welcome screen will list the current version being run and will await further commands after the R prompt “>”

$ R

R version 3.0.3 (2014-03-06) -- "Warm Puppy"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (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.

Natural language support but running in an English locale

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.

> 

Note: If you are using R from a local installation e.g. on a Windows or Mac, simply locate the R icon in your system and double-click it:

Macintosh: R is installed in the Applications directory. It will appear as R or R.app depending on your viewing options.

Windows: R most likely installed a shortcut on the desktop. Otherwise search within the Windows Start button.
R objects

R keeps information in RAM in the form of “R objects” which can be thought of as a “container of information” just like a vase can contain water, and a box contain cookies, chocolates or utensils. In some cases the box could have separators so that the cookies don’t stick to each other… in the same way R objects may have “structure” that organizes the data in a meaningful and useful way for later retrieval.

The name of an object must start with a letter (A–Z and a–z) but can include letters, digits (0–9), dots (.), and underscores (_). R discriminates between uppercase letters and lowercase ones in the names of the objects, so that x and X can name two distinct objects (even under Windows).

1. Simplest, implicit command

One of the simplest commands is to type the name of an object to display its content. For instance, if an object n contains the value 15:

> n
[1] 15

The digit [1] within brackets indicates that the display starts at the first element of n. This command is an implicit use of the function print() and the above example is similar to print(n).
2. The “assign” operator (= or <-) : create, list and delete object in memory

An object can be created with the “assign” operator which is written as an arrow created with a minus sign and a less-than or greater than symbol (<- or ->); this symbol can be oriented left-to-right or the reverse: In most cases the equal sign (=) can also be used:

(reminder note: user’s input is in bold letters)

> n <- 15

> n
[1] 15

> 5 -> n

> n
[1] 5

If the object already exists, its previous value is erased (the modification affects only the objects in the active memory, not the data on the disk). Therefore the value 15 contained within n was replaced by 5.

The value assigned this way may be the result of an operation and/or a function:

> n <- 10 + 2

> n
[1] 12

The following lines illustrates that R is case sensitive:

> x = 1

> X = 10

> x
[1] 1

> X
[1] 10

Note that you can simply type and calculate an expression without assigning its value to an object.

The result is thus displayed immediately on the screen and is not stored in memory:
R objects

> (10 + 2) * 5
[1] 60
R can therefore be used as a calculator:

> 2 + 2
[1] 4

> sqrt(10)
[1] 3.162278

> 2*3*4
[1] 24

> 3^2
[1] 9

> 2^16
[1] 65536

> exp(1)
[1] 2.718282 # value of “e”

> log(10)
# natural log
[1] 2.302585

> log10(1000)
# log base 10
[1] 3

> pi
[1] 3.141593

> sin(30*pi/180)
# convert angles to radians and then applies the sinus function
[1] 0.5

> n <- 15

> 4*n
[1] 60

Note: In R, in order to be executed, a function always needs to be written with parentheses, even if there is nothing within them e.g. `ls()`. If one just types the name of a function without parentheses, R will display the content of the function instead.
The semi-colon (;) can be used to separate distinct commands on the same line:

```r
> name <- "Carmen"; n1 <- 10; n2 <- 100; m <- 0.5
```

The function `ls()` simply lists the R objects currently in memory: only the names of the objects are displayed:

```r
> ls()
[1] "m"  "n1"  "n2"  "name"
```

(Note: if you typed `n <- 15` in the above section, there will also be “n” listed here)

If there are a large number of objects in memory, it may be useful to list only those of interest, for example those containing the letter “m” within their name. In a Windows DOS command that could be done with `C> DIR *m*` while in Unix it could be done with `% ls *m*`. Within R the search pattern (option `pattern` is abbreviated `pat`) is placed within the parentheses and there is no need for the wild card (*). This is how we will look for the pattern `m`:

```r
> ls(pat = "m")
[1] "m"  "name"
```

To restrict the search to objects that start with the letter `m` (in technical term this is called a “regular expression”):

```r
> ls(pat = "^m")
[1] "m"
```

To delete objects in memory, we use the function `rm`:

- `rm(x)` deletes the object `x`,
- `rm(x, y)` deletes both the objects `x` and `y`,
- `rm(list=ls())` deletes all the objects in memory;

The same options mentioned for the function `ls()` can then be used to delete selectively some objects: `rm(list=ls(pat="^m")).`

### 3. Online help

Help pages are accessed with the simple commands `?` or `help()`. For example the following two commands have the same effect:

```r
> ?ls

> help(ls)
```
The help page may appear within the R console or within a separate window depending on the version and operating system.

Note that the functions usually have a series of optional parameters that have a default. For example the function `ls()` has the following definition of which we already know “pattern” from the above example:

```r
text: \texttt{ls(name, pos = -1, envir = as.environment(pos), all.names = FALSE, pattern)}
```

For functions that contain special characters, it is necessary to use quotes:

```r
> ?"*"

> help("*")
```
Data with R

R can manipulate numbers and words (“strings” in programing language). R Objects can contain this information in various forms. This is what is explained further below.

1. R Objects

R works with objects, which are characterized by their name and content. Objects have also an attribute that specifies which kind of data is represented by an object. All objects have two intrinsic attributes: mode and length. The mode is the basic type of the elements contained within the object; there are four main modes: numeric, character, complex and logical (FALSE or TRUE). The length is the number of elements of the object. The functions mode() and length() are used to display the mode and length of an object.

Example: (user’s input in bold character) also making use of the semi-colon separator as we already learned above:

> x <- 1

> mode(x)
[1] "numeric"

> length(x)
[1] 1

> A <- "bacteria"; compar <- TRUE; z <- 1i

> mode(A); mode(compar); mode(z)
[1] "character"
[1] "logical"
[1] "complex"

> length(A); length(compar); length(z)
[1] 1
Note that the length is not representing the number of letters in a word.

- Whatever the mode, missing data are represented with \texttt{NA} (not available).
- Values that are not numbers are represented with \texttt{NaN} (not a number).
- Infinity is represented with \texttt{Inf} and \texttt{–Inf}.

A value of mode character is input with single or double quotes. The echo is always double quotes.

\begin{verbatim}
> A <- "bacteria"
> B <- 'E.coli'
> A; B
\end{verbatim}

The backslash can be used to “escape” a special character. The two characters altogether \texttt{\} will be treated in a specific way by some functions such as \texttt{cat} for display on screen:

\begin{verbatim}
> x <- "Double quotes \ by delimitate R's strings."
> x
[1] "Double quotes \ by delimitate R's strings."
> cat(x)
Double quotes " by delimitate R's strings.
\end{verbatim}

The following table gives an overview of the type of objects representing data.

\begin{table}[ht]
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{object} & \textbf{modes} & \textbf{several modes possible in the same object?} \\
\hline
\texttt{vector} & numeric, character, complex or logical & No \\
\texttt{factor} & numeric or character & No \\
\texttt{array} & numeric, character, complex or logical & No \\
\texttt{matrix} & numeric, character, complex or logical & No \\
\texttt{data frame} & numeric, character, complex or logical & Yes \\
\texttt{list} & numeric, character, logical, function, expression, … & Yes \\
\hline
\end{tabular}
\end{table}
A vector is a variable in the commonly admitted meaning.

A factor is a categorical variable.

An array is a table with k dimensions, a matrix being a particular case of array with k = 2. Note that the elements of an array or of a matrix are all of the same mode.

A data frame is a table composed with one or several vectors and/or factors all of the same length but possibly of different modes.

A ‘ts’ is a time series data set and so contains additional attributes such as frequency and dates.

Finally, a list can contain any type of object, included lists!

For a vector, its mode and length are sufficient to describe the data. For other objects, other information is necessary and it is given by non-intrinsic attributes. Among these attributes, we can cite dim which corresponds to the dimensions of an object. For example, a matrix with 2 lines and 2 columns has for dim the pair of values [2, 2], but its length is 4.

2. Reading data from a file

When R is first started, the software will “look” into the default directory also referred to as the working directory. For reading and writing in files, R uses the working directory.

By default this will be the “home” directory of the user. For SBGrid users this will likely be the default $HOME defined variable, for example on a Macintosh: /Users/user1

To find this directory, the command getwd() (get working directory) can be used, and the working directory can be changed with setwd("C:/data") or setwd("/home/~paradis/R").

Important: It is necessary to give the path to a file if it is not in the working directory.

On the Windows and Mac systems installed as stand-alone applications the working directory can be changed with one of the pull-down menu thanks to the graphical interface, which is different on the 2 platforms:
Data with R

Note that this is not available on the SBGrid session running within the Terminal.

The following R functions can read data stored in plain text format (ASCII): 
- `read.table()` (there are several variants, shown below),
- `scan` and
- `read.fwf()` (read fixed width format). These functions are part of the R `base` package. Other packages offer functions to read files from Excel or other statistical packages and only useful for more advanced R sessions (not shown here.)

The function `read.table()` creates a data frame (see definition above) when the file is read.

For instance, if one has a file named data.dat, the command:

```r
> mydata <- read.table("data.dat")
```

will create a data frame named `mydata`, and each variable will be named, by default, `V1`, `V2`, ..., and can be accessed individually by `mydata$V1`, `mydata$V2`, ..., or by `mydata["V1"]`, `mydata["V2"]`, ..., or, still another solution, by `mydata[, 1]`, `mydata[,2]`, ... However, there is a difference: `mydata$V1` and `mydata[, 1]` are vectors whereas `mydata["V1"]` is a data frame. We shall see later how to manipulate objects.

There are several options whose default values (i.e. those used by R if they are omitted by the user) are detailed in the following table:

```
read.table(file, header = FALSE, sep = " ", quote = "\" ",
dec = ",", row.names, col.names,
as.is = !stringsAsFactors,
aa.strings = "NA", colClasses = NA, nrows = -1,
skip = 0, check.names = TRUE, fill = !blank.lines.skip,
strip.white = FALSE, blank.lines.skip = TRUE,
comment.char = "#", allowEscapes = FALSE, flush = FALSE,
stringsAsFactors = default.stringsAsFactors())
```
Saving data into a file

<table>
<thead>
<tr>
<th>file</th>
<th>the name of the file to be opened (within quotes”). \ symbol is not allowed even under Windows and must be replaced by /</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>a logical (FALSE or TRUE) indicating if the file contains the name of the variables on its first line.</td>
</tr>
<tr>
<td>sep</td>
<td>field separator used in the file. For instance, TAB-delimited tabulation: sep = “\t”</td>
</tr>
<tr>
<td>quote</td>
<td>the character used to cite the variables of mode character</td>
</tr>
<tr>
<td>dec</td>
<td>the character used for decimal point</td>
</tr>
<tr>
<td>row.names</td>
<td>a vector or row names. If row.names is missing, the rows are numbered. Using row.names = NULL forces row numbering.</td>
</tr>
<tr>
<td>col.names</td>
<td>a vector with the names of the variables (by default V1, V2, V3 …)</td>
</tr>
<tr>
<td>nrows</td>
<td>the maximum number of rows to read in. Negative values are ignored.</td>
</tr>
<tr>
<td>skip</td>
<td>the number of lines of the data file to skip before beginning to read data.</td>
</tr>
<tr>
<td>fill</td>
<td>Logical. If TRUE then in case the rows have unequal length, blank fields are implicitly added.</td>
</tr>
</tbody>
</table>

The complete description of all the parameters are in the help file:

```r
> help(read.table)
```

The `read.table` variants differ in the default values of some of the parameters:

**Comma delimited text:**

```r
read.csv(file, header = TRUE, sep = "","", quote="\"", dec=".", fill = TRUE, comment.char="", ...)  
read.csv2(file, header = TRUE, sep = ";","", quote="\"", dec="", fill = TRUE, comment.char="", ...)  
```

**Tab delimited text:**

```r
read.delim(file, header = TRUE, sep = "\t", quote="\"", dec=".", fill = TRUE, comment.char="", ...)  
read.delim2(file, header = TRUE, sep = "\t", quote="\"", dec="", fill = TRUE, comment.char="", ...)  
```

3. Saving data into a file

The function `write.table` writes into a file an object, typically a data frame but this could well be another kind of object (vector, matrix, …). The arguments and options are:
To write in a simpler way an object in a file, the command `write(x, file="data.txt")` can be used, where `x` is the name of the object (which can be a vector, a matrix, or an array). There are two options: `nc` (or `ncol`) which defines the number of columns in the file (by default `nc=1` if `x` is of mode character, `nc=5` for the other modes), and `append` (a logical) to add the data without deleting those possibly already in the file (TRUE) or deleting them if the file already exists (FALSE, the default).

To record a group of objects of any type, we can use the command `save(x, y, z, file= "xyz.RData").` To ease the transfert of data between different machines, the option `ascii = TRUE` can be used. The data (which are now called a workspace in R's jargon) can be loaded later in memory with `load("xyz.RData").` The function `save.image()` is a short-cut for `save(list =ls(all=TRUE), file=".RData").`
will generate an object with 30 elements; a regular sequence of integers ranging from 1 to 30:

```
> x
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
[16] 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

The operator “:`” has priority over the arithmetic operators

```
> 1:10-1
[1]  0  1  2  3  4  5  6  7  8  9

> (1:10)-1
[1]  0  1  2  3  4  5  6  7  8  9

> 1:(10-1)
[1]  1  2  3  4  5  6  7  8  9

> 1:10-0.1
[1]  0.9  1.9  2.9  3.9  4.9  5.9  6.9  7.9  8.9  9.9
```

The function `seq()` can also generate real numbers series:

```
> seq(1, 5, 0.4)
[1] 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0
```

or alternatively:

```
> seq(length=11, from=1, to=5)
[1] 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0
```

where the first number indicates the beginning of the sequence, the second one the end, and the third one the increment to be used to generate the sequence. One can also type the values directly with the **c()** function:

```
> c(1.0, 1.4, 1.8, 2.2, 2.6, 3.0, 3.4, 3.8, 4.2, 4.6, 5.0)
[1] 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0
```

**Note:** the **c()** function is used very often to type explicit data within the input of other functions that combines its arguments to form a vector.

It is also possible, if one wants to enter some data on the keyboard, to use the function **scan()** with simply the default options:

```
> z <- scan()
1: 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0
```
> z
[1] 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0

The function `rep()` creates a vector with all its elements identical:

> rep(1, 20)
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

The function `sequence` creates a series of sequences of integers each ending by the numbers given as arguments (underline added for clarity)

> sequence(2:5)
[1] 1 2 1 2 3 1 2 3 4 1 2 3 4 5

> sequence(c(2,5))
[1] 1 2 1 2 3 4 5

The function `gl()` (generate levels) is very useful because it generates regular series of factors. The usage of this function is `gl(k, n)` where `k` is the number of levels (or classes), and `n` is the number of replications in each level. Two options may be used: `length` to specify the number of data produced, and `labels` to specify the names of the levels of the factor. Examples:

> gl(3, 5)
[1] 1 1 1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3

> gl(3, 5, length=30)
[1] 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 1 1 1 1 1 1 2 2 2 2 2 3 3 3 3
Levels: 1 2 3

> gl(2, 6, label=c("Male", "Female"))
[7] Female Female Female Female Female Female
Levels: Male Female

> gl(2, 10)
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
Levels: 1 2

> gl(2, 1, length=20)
[1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Levels: 1 2
> gl(2, 2, length=20)
  [1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
Levels: 1 2

Finally, `expand.grid()` creates a data frame with all possible combinations of vectors or factors given as arguments: (Note the extensive use of the `c()` function for each argument!)

> expand.grid(h=c(60,80), w=c(100, 300),
              sex=c("Male", "Female"))
   h  w   sex
1  60 100 Male
2  80 100 Male
3  60 300 Male
4  80 300 Male
5  60 100 Female
6  80 100 Female
7  60 300 Female
8  80 300 Female

> expand.grid(myX=c(1,2), myY=c(10, 20), Case=c("A", "B", "C"))
      myX myY Case
1      1  10   A
2      2  10   A
3      1  20   A
4      2  20   A
5      1  10   B
6      2  10   B
7      1  20   B
8      2  20   B
9      1  10   C
10     2  10   C
11     1  20   C
12     2  20   C

Note: the number of combination is the multiplication of the number of arguments, here 2 x 2 x 3 = 12 cases.

4.2. Random sequences

Most of the statistical functions are available within R such as Gaussian (Normal), Poisson, Student t test etc.

Example for the Gaussian function:
> help(dnorm)

Note: this is a hypertext illustration in Windows.

This help command within an SBGrid terminal will show the same information in plain text within the terminal.

Type the letter q to quit the plain text display or the space-bar to display the next screenfull.

Graphical illustration of the distribution functions.
The first graph (dnorm) was obtained with the command:

> plot(function(x) dnorm(x), -5, 5)

The normal function is abbreviated “norm” with one of the added prefix: d, p, q or r meaning density, distribution, quantile and random respectively: dnorm, pnorm, qnorm and rnorm.

To generate random numbers, the function rnorm() can be used. The number of desired random numbers is given as argument.

Since these are random, the answers are never the same:

> rnorm(1)
[1] 0.01160411

> rnorm(1)
[1] 0.1730448

Note that with rnorm() the val-ues are different each time!
Generating data

> \texttt{rnorm(2)}
> [1] 0.83653193 -0.06752702

The number in parentheses indicates how many random numbers we want to generate.

> \texttt{rnorm(2)}
> [1] 0.4218784 -0.7225086

> \texttt{rnorm(2)}
> [1] 0.7537601 1.2409371

Example: calculate 5 random numbers using variable x:

> \texttt{x <- 1:5}
> \texttt{x}
> [1] 1 2 3 4 5

> \texttt{rnorm(x)}
> [1] -0.93522503 -1.02403529 -0.28424994 -0.38654353
> [5] -1.16811404

The list of functions to generate random sequence is shown in this table:

<table>
<thead>
<tr>
<th>law</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian (normal)</td>
<td>\texttt{rnorm(n, mean=0, sd=1)}</td>
</tr>
<tr>
<td>exponential</td>
<td>\texttt{rexp(n, rate=1)}</td>
</tr>
<tr>
<td>gamma</td>
<td>\texttt{rgamma(n, shape, scale=1)}</td>
</tr>
<tr>
<td>Poisson</td>
<td>\texttt{rpois(n, lambda)}</td>
</tr>
<tr>
<td>Weibull</td>
<td>\texttt{rweibull(n, shape, scale=1)}</td>
</tr>
<tr>
<td>Cauchy</td>
<td>\texttt{rcauchy(n, location=0, scale=1)}</td>
</tr>
<tr>
<td>beta</td>
<td>\texttt{rbeta(n, shape1, shape2)}</td>
</tr>
<tr>
<td>Student’ (t)</td>
<td>\texttt{rt(n, df)}</td>
</tr>
<tr>
<td>Fisher–Snedecor (F)</td>
<td>\texttt{rf(n, df1, df2)}</td>
</tr>
<tr>
<td>Pearson (\chi^2)</td>
<td>\texttt{rchisq(n, df)}</td>
</tr>
<tr>
<td>binomial</td>
<td>\texttt{rbinom(n, size, prob)}</td>
</tr>
<tr>
<td>multinomial</td>
<td>\texttt{rmultinom(n, size, prob)}</td>
</tr>
<tr>
<td>geometric</td>
<td>\texttt{rgeom(n, prob)}</td>
</tr>
<tr>
<td>hypergeometric</td>
<td>\texttt{rhyper(nn, m, n, k)}</td>
</tr>
<tr>
<td>logistic</td>
<td>\texttt{rlogis(n, location=0, scale=1)}</td>
</tr>
<tr>
<td>lognormal</td>
<td>\texttt{rlnorm(n, meanlog=0, sdlog=1)}</td>
</tr>
<tr>
<td>negative binomial</td>
<td>\texttt{rnbinom(n, size, prob)}</td>
</tr>
<tr>
<td>uniform</td>
<td>\texttt{runif(n, min=0, max=1)}</td>
</tr>
<tr>
<td>Wilcoxon’s statistics</td>
<td>\texttt{rwilcox(nn, m, n)}, \texttt{rsignrank(nn, n)}</td>
</tr>
</tbody>
</table>
5. Manipulating objects

Section 3.5 of the Emmanuel Paradis’s “R for Beginners” (pages 18 – 35) is 18 pages long and the reader is encouraged to review these pages (reminder download from: cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf).

Methods for accessing objects values by indexing will be reviewed here.

Accessing and changing the value within a simple number vector:

```r
> x <- 0:5
> x
[1] 0 1 2 3 4 5
> x[3]
[1] 2
> x[3] <- 100
> x
[1] 0 1 100 3 4 5
```

If `x` is a matrix or a data frame, the value of the `i`th line and `j`th column is accessed with `x[i, j]`.

To access all values of a given row or column, one has simply to omit the appropriate index (without forgetting the comma):

```r
> x <- matrix(1:6, 2, 3)
> x
[,1] [,2] [,3]
[1,]  1  3  5
[2,]  2  4  6
> x[1,]
[1] 1 3 5
> x[,1]
[1] 1 2
> x[1,1]
[1] 1
```

The value of any column, row or single value can be changed by simply assigning new values:

```r
> x[, 3] <- 21:22
> x
```

The new values of `x` are displayed.
You have certainly noticed that the last result is a vector and not a matrix. The default behavior of R is to return an object of the lowest dimension possible. This can be altered with the option `drop` which default is `TRUE`:

```r
> x[, 3, drop = FALSE]
 [,1] [,2]
[1,]  21  22
[2,]  21  22
```

Accessing or printing subsets:

```r
> z <- matrix(1:30, 5, 6)
> z
[1,]   1   6  11  16  21  26
[2,]   2   7  12  17  22  27
[3,]   3   8  13  18  23  28
[4,]   4   9  14  19  24  29
[5,]   5  10  15  20  25  30
```

First create a matrix containing numbers 1 through 30 (1:30) named `z` made of 5 rows of 6 columns.

Then show the content of the matrix.

```r
> z[3:4, 4:5]
 [,1] [,2]
[1,]  16  17
[2,]  22  23
```

Finally, write out a subset of the large matrix from 3rd row and 4th column to 4th row and 5th column shown as bold+underline above.

Note that the columns and rows are renumbered 1 and 2.
This indexing system is easily generalized to arrays, with as many indices as the number of dimensions of the array. Example for a three dimensional array:

\[ x[i, j, k], x[, , 3], x[, , 3, \text{drop} = \text{FALSE}], \text{and so on}. \]

In some cases, it may be very useful to bind or “glue” 2 matrices or data tables together. The functions \( \text{rbind()} \) and \( \text{cbind()} \) can bind matrices with respect to lines or columns respectively:

```r
> m1 <- matrix(1, nr = 2, nc = 2)
> m1
   [,1] [,2]
[1,]  1  1
[2,]  1  1
> m2 <- matrix(2, nr = 2, nc = 2)
> m2
   [,1] [,2]
[1,]  2  2
[2,]  2  2
> cbind(m1, m2)
[1,]  1  1  2  2
[2,]  1  1  2  2
> rbind(m1, m2)
   [,1] [,2]
[1,]  1  1
[2,]  1  1
[3,]  2  2
[4,]  2  2
```

Matrix \( m1 \) is created to contain the digit 1 in all rows and columns. There are 2 rows (\( \text{nr} = \text{number of rows} \)) and 2 columns (\( \text{nc} = \text{number of columns} \)).

Matrix \( m2 \) is created in a similar manner with the value 2.

\( \text{cbind()} \) is used to bind (glue) the 2 matrices next to each other. It is implied that the number of rows is identical.

\( \text{rbind()} \) is used to collate the matrices above each other. It is implied that the number of columns is identical.

Let’s introduce matrix \( m3 \) to test the assumptions of equal number of rows or columns: \( m3 \) contains 2 columns but 3 rows.

```r
> m3 <- matrix(3, nc=2, nr=3)
> m3
   [,1] [,2]
[1,]  3  3
[2,]  3  3
[3,]  3  3
> cbind(m1, m3)
Error in cbind(deparse.level, ...) : number of rows of matrices must match (see arg 2)
```

Matrix \( m3 \) contains 2 columns but 3 rows.

Therefore the \( \text{cbind()} \) function cannot be used on the entire matrix.
However it can be used if the some rows are eliminated. Since all numbers are the value 3, the subset 
\texttt{m3[2:3,1:2]} would provide the same result.

In the case of \texttt{m2} and \texttt{m3} since they have the same number of columns we can use the \texttt{rbind()} function to assemble them:

\begin{verbatim}
> rbind(m1, m3)
[,1] [,2]
[1,]  1  1
[2,]  1  1
[3,]  3  3
[4,]  3  3
[5,]  3  3
\end{verbatim}

\begin{verbatim}
> m3[2:3,1:2]
[,1] [,2]
[1,]  3  3
[2,]  3  3
\end{verbatim}

\texttt{rbind()} in this case works because the number of columns is identical.
Section 4 of the Emmanuel Paradis’s “R for Beginners” (pages 36 – 54) is a 19 pages segment covering many aspects of graphics.

The following mini exercise will be useful to understand later plots:

First create a list of 1000 points, and display the first 10 and last 10 of the series.

```r
> x <- 1:1000

> x[1:10] ; x[990:1000]
[1]  1  2  3  4  5  6  7  8  9 10
[1]  990 991 992 993 994 995 996 997 998 999 1000

> data <- rnorm(x)

> plot(data)
> abline(h=2)
> abline(h=0)
> abline(h=-2)

> above2 <- data > 2

> sum(above2)
[1] 23

> above2[35:39]
[1] FALSE FALSE TRUE FALSE FALSE FALSE

> below_2 <- data < (-2)
```

Do the same calculations for points below –2.
Manipulating objects

> \texttt{sum(below\_2)}
> [1] 24

Note that \texttt{below\_2} is a valid vector name but \texttt{below-2} is not! There are 24 points that satisfy this condition.

> \texttt{points(x[above2], data[above2], pch=20, col="red")}

Replot points with specific colors.

> \texttt{points(x[below\_2],data[below\_2],pch=20, col="blue")}

Points above 2 are colored red.

Points in the middle were not changed.

Points less than \texttt{-2} are colored blue.
1. Plotting symbols

Here is a table of many available symbols as described in E. Paradis’s manual “R for beginners”

(English version, page 44. See credentials above.)

![Plotting symbols in R (pch=1:25). The colours were obtained with the options col="blue", bg="yellow", the second option has an effect only for the symbols 21-25. Any character can be used (pch="*", "?", ",", ",", ...).]

You can try the following example code\(^2\) that shows all 25 symbols that can be used to produce points in graphs:

```r
# Make an empty chart
plot(1, 1, xlim=c(1,5.5), ylim=c(0,7), type="n", ann=FALSE)

# Plot digits 0-4 with increasing size and color
text(1:5, rep(6,5), labels=c(0:4), cex=1:5, col=1:5)
```

\(^2\) From: bottom of page at: http://www.harding.edu/fmccown/R/
# Plot symbols 0-4 with increasing size and color
points(1:5, rep(5,5), cex=1:5, col=1:5, pch=0:4)
text((1:5)+0.4, rep(5,5), cex=0.6, (0:4))

# Plot symbols 5-9 with labels
points(1:5, rep(4,5), cex=2, pch=(5:9))
text((1:5)+0.4, rep(4,5), cex=0.6, (5:9))

# Plot symbols 10-14 with labels
points(1:5, rep(3,5), cex=2, pch=(10:14))
text((1:5)+0.4, rep(3,5), cex=0.6, (10:14))

# Plot symbols 15-19 with labels
points(1:5, rep(2,5), cex=2, pch=(15:19))
text((1:5)+0.4, rep(2,5), cex=0.6, (15:19))

# Plot symbols 20-25 with labels
points((1:6)*0.8+0.2, rep(1,6), cex=2, pch=(20:25))
text((1:6)*0.8+0.5, rep(1,6), cex=0.6, (20:25))

Notes:

cex determines the size of the plotted pch symbol.

rep(n,5) repeats the n value for plotting 5 times as a horizontal line. In essence that is the y coordinate for the point to be plotted.

In truth, since numbers and letters can be used for plotting there are over 100 characters that can be used to plot.

Using what we know about matrices and the example code above we can write:

# Create a Matrix Mt containing 12 rows of numbers
# from 0 to 143, filled by row
> Mt<- matrix (c(0:143), ncol=12,byrow=TRUE)

> Mt

```
[1,]  0   1   2   3   4   5   6   7   8   9  10  11
[2,] 12  13  14  15  16  17  18  19  20  21  22  23
[3,] 24  25  26  27  28  29  30  31  32  33  34  35
[4,] 36  37  38  39  40  41  42  43  44  45  46  47
[5,] 48  49  50  51  52  53  54  55  56  57  58  59
[6,] 60  61  62  63  64  65  66  67  68  69  70  71
[7,] 72  73  74  75  76  77  78  79  80  81  82  83
[8,] 84  85  86  87  88  89  90  91  92  93  94  95
[9,] 96  97  98  99 100 101 102 103 104 105 106 107
[10,]108 109 110 111 112 113 114 115 116 117 118 119
[11,]120 121 122 123 124 125 126 127 128 129 130 131
```
The command could easily be altered to print everything in blue with the few objects with a fill in yellow by altering the line with the `points` command:

```r
points((1:12), rep(i, 12), pch=Mt[i+1,], col="blue", bg="yellow")
```

It is to be noted that there are no symbols for values ranging from 25 to 31 and the plot appears blank for these coordinates.

### 2. Split screen multiple plots

The parameters `mfrow` and `mfcol` can be used to split the plotting surface into specified numbers of rows and columns respectively.

The combinations can be quite complex (see Paradis’s tutorial for that) and the number of parameters also quite large.

Type `?par` to see the help file and the list of parameters!
Here is the help definition for those 2 parameters

mfcol, mfrow
A vector of the form c(nr, nc). Subsequent figures will be drawn in an nr-by-nc array on the device by columns (mfcol), or rows (mfrow), respectively.

Typically it is the \texttt{par(mfrow = c(nr, nc))} version that is more often used.

\textbf{Note that the screen will remain “split” until specified otherwise!}

The following example \textsuperscript{3} helps understand this. First we create a series of 50 random numbers that are plotted in 4 separate methods (\texttt{type=}) on a split screen (\texttt{mfrow}):

```r
> x1 <- rnorm(50)
> par(mfrow = c(2,2))
> plot(x1, type = "p", main = "points", ylab = "", xlab = "")
> plot(x1, type = "l", main = "lines", ylab = "", xlab = "")
> plot(x1, type = "b", main = "both", ylab = "", xlab = "")
> plot(x1, type = "o", main = "both overplot", ylab = "", xlab = ")
> par(mfrow = c(1,1))
```

\textsuperscript{3} From http://en.wikibooks.org/wiki/R_Programming/Graphics
End Hands On Tutorial
Appendix A: R outside SBGrid

The Comprehensive R Archive Network

Go to: www.r-project.org

Click download R on the main page and choose a mirror near you

Select the type of computer (Linux, Mac, Windows) and download the installer.

Installation is guided by installer software.