# A (very) short introduction to R

## 1 Introduction

R is a powerful language and environment for statistical computing and graphics. It is a public domain (a so called GNU) project which is similar to the commercial S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S, and is much used in as an educational language and research tool. The main advantages of R are the fact that R is freeware and that there is a lot of help available online. It is quite similar to other programming packages such as MatLab (not freeware), but more user-friendly than programming languages such as C++ or Fortran. You can use R as it is, but for educational purposes we prefer to use R in combination with the Rstudio interface (also freeware), which has an organized layout and several extra options.
The aim of these workstation sessions is for you to experience using the course software, R, for yourself. They will allow you to develop the necessary skills required to undertake the sorts of practical analyses which will be presented to you in lectures by working through the exercises provided, but will also give you the opportunity to explore further what this software has to offer.

## 2 Getting started

### 2.1 Install R

To install R on your computer (legally for free!), go to the home website of R:

`http://www.r-project.org/`

and do the following (assuming you work on a windows computer):

- click download CRAN in the left bar
- choose a download site
- choose Windows as target operation system
- click base
- choose Download R for Windows and choose default answers for all questions

## 2.2 Install RStudion

After finishing this setup, you should see an R icon on you desktop. Clicking on this would start up the standard interface. We recommend, however, to use the Rstudio interface. To install Rstudio, go to:

`http://www.rstudio.org/`

and do the following (assuming you work on a win- dows computer):

- click Download Rstudio
- click Recommended For Your System
- download the .exe file and run it (choose default answers for all questions)

## 2.3 Rstudio layout

The Rstudio interface consists of several windows

- Bottom left: console window (also called command window). Here you can type simple commands after the ">" prompt and R will then execute your command. This is the most important window, because this is where R actually does stuff.
- Top left: editor window (also called script window). Collections of commands (scripts) can be edited and saved. Just typing a command in the editor window is not enough, it has to get into the command window before R executes the command. If you want to run a line from the script window (or the whole script), you can click `Run` or press `CTRL+ENTER` to send it to the command window.
- Top right: workspace/history window. In the workspace window you can see which data and values R has in its memory. You can view and edit the values by clicking on them. The history window shows what has been typed before.
- Bottom right: files/plots/packages/help window. Here you can open files, view plots (also previous plots), install and load packages or use the help function.

## 2.4 Working directory

Your working directory is the folder on your computer in which you are currently working. When you ask R to open a certain file, it will look in the working directory for this file, and when you tell R to save a data file or figure, it will save it in the working directory. Before you start working, please set your working directory to where all your data and script files are or should be stored. Type in the command window: `setwd("directoryname")`. For example:

```
> setwd("M:/Lab/R/")
```

Make sure that the slashes are forward slashes and that you don't forget the apostrophes. R is case sensitive, so make sure you write capitals where necessary.

## 2.5 Libraries

R can do many statistical and data analyses. They are organized in so-called packages or libraries. With the standard installation, most common packages are installed. To get a list of all installed packages, go to the packages window or type `library()` in the console window. If the box in front of the package name is ticked, the package is loaded (activated) and can be used. There are many more packages available on the R website. If you want to install and use a package (for example, the package called `"geometry"`) you should:

- Install the package: click install packages in the packages window and type geometry or type `install.packages("geometry")` in the command window.
- Load the package: check box in front of `geometry` or type `library("geometry")` in the command window.

# 3 Some first examples of R commands

## 3.1 Calculator

R can be used as a calculator. You can just type your equation in the command window after the ">":

```
> 10^3 + 36
```

and R will give the answer

```
[1] 1036
```

## 3.2 Workspace

You can also give numbers a name. By doing so, they become so-called variables which can be used later. For example, you can type in the command window:

```
> a = 10
```

You can see that a appears in the workspace window, which means that R now remembers what `a` is. You can also ask R what `a` is (just type `a ENTER` in the command window):

```
> a
[1] 10
```

or do calculations with `a`:

```
> a * 7
[1] 70
```

If you specify `a` again, it will forget what value it had before. You can also assign a new value to `a` using the old one.

```
> a = a + 15
> a
[1] 25
```

To remove all variables from R's memory, type

```
> rm(list=ls())
```

or click "clear" icon in the workspace window.

## 3.3 Scalars, vectors and matrices

Like in many other programs, R organizes numbers in scalars, vectors and matrices. The `a` you defined before was a scalar. To define a vector with the numbers 5, 6 and 7, you need the function `c`, which is short for concatenate.

```
> b=c(5,6,7)
```

## 3.4 Functions

If you would like to compute the mean of all the elements in the vector `b` from the example above, you could type

```
> (5+6+7)/3
```

But when the vector is very long, this is very boring and time-consuming work. This is why things you do often are automated in so-called functions. Some functions are standard in R or in one of the packages. You can also program your own functions. When you use a function to compute a mean, you'll type:

```
> mean(b)
```

The function `rnorm`, as another example, is a standard R function which creates random samples from a normal distribution. Hit the `ENTER` key and you will see 10 random numbers as:

```
> rnorm(10)
[1]  0.6589677 -0.6912087  1.1978609 -0.5968915 -2.0587241 -1.1309952  0.7023617
[8] -0.6676629  2.0607799  0.2769710
```

The First line contains the command: `rnorm` is the function and the 10 is an argument specifying how many random numbers you want - in this case 10 numbers (typing `n=10` instead of just 10 would also work). Lines 2-3 contain the results: 10 random numbers organised in a vector with length 10. Entering the same command again produces 10 new random numbers. Instead of typing the same text again, you can also press the upward arrow key (↑) to access previous commands. If you want 10 random numbers out of normal distribution with mean 1.2 and standard deviation 3.4 you can type

```
> rnorm(10, mean=1.2, sd=3.4)
```

showing that the same function (rnorm) may have different interfaces and that R has so called named arguments (in this case `mean` and `sd`). Comparing this example to the previous one also shows that for the function `rnorm` only the first argument (the number 10) is compulsory, and that R gives default values to the other so-called optional arguments.

## 3.5 Plots

R can make graphs. The following is a very simple example:

```
> x = rnorm(100)
> plot(x)
```

In the first line, 100 random numbers are assigned to the variable `x`, which becomes a vector by this operation. In the second line, all these values are plotted in the plots window.

## 4 Help and documentation

There is a large amount of (free) documentation and help available. Some help is automatically installed. Typing in the console window the command

```
> help(rnorm)
```

gives help on the `rnorm` function. It gives a description of the function, possible arguments and the values that are used as default for optional arguments. Typing

```
> example(rnorm)
```

gives some examples of how the function can be used. An HTML-based global help can be called with:

```
> help.start()
```

or by going to the help window. The following links can also be very useful:

- `https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf` A full manual.
- `http://cran.r-project.org/doc/contrib/ Short-refcard.pdf` A short reference card.
- `http://www.statmethods.net/` Also called Quick-R. Gives very productive direct help. Also for users coming from other programming languages.
- `http://mathesaurus.sourceforge.net/` Dictionary for programming languages (e.g. R for Matlab users).
- Just using Google (type e.g. "R rnorm" in the search field) can also be very productive.

## 5 Scripts

R is an interpreter that uses a command line based environment. This means that you have to type commands, rather than use the mouse and menus. This has the advantage that you do not always have to retype all commands.
You can store your commands in files, the socalled scripts. These scripts have typically file names with the extension `.R`, e.g. `foo.R`. You can open an editor window to edit these files by clicking `File` and `New` or `Open file`. You can run (send to the console window) part of the code by selecting lines and clicking `Run` in the editor window. If you do not select anything, R will run the line your cursor is on. You can always run the whole script with the console command source, so e.g. for the script in the file `foo.R` you type:

```
> source("foo.R")
```

You can also click `Run` all in the editor window to run the whole script at once.

## 6 Data structures

### 6.1 Vectors

Vectors were already introduced, but they can do more:

```
> vec1 = c(1,4,6,8,10)
> vec1
[1] 1 4 6 8 10
> vec1[5]
[1] 10
> vec1[3] = 12
> vec1
[1] 1 4 12 8 10
> vec2 = seq(from=0, to=1, by=0.25)
> vec2
[1] 0.00 0.25 0.50 0.75 1.00
> sum(vec1)
[1] 35
> vec1 + vec2
[1] 1.00 4.25 12.50 8.75 11.00
```

- In line 1, a vector `vec1` is explicitly constructed by the concatenation function `c()`, which was introduced before. Elements in vectors can be addressed by standard `[i]` indexing, as shown in lines 4-5.
- In line 6, one of the elements is replaced with a new number. The result is shown in line 8.
- Line 9 demonstrates another useful way of constructing a vector: the `seq()` (sequence) function.
- Lines 10-15 show some typical vector oriented calculations. If you add up two vectors of the same length, the first elements of both vectors are summed, and the second elements, etc., leading to a new vector of length 5 (just like in regular vector calculus). Note that the function `sum` sums up the elements within a vector, leading to one number (a scalar).

### 6.2 Matrices

Matrices are nothing more than 2-dimensional vectors. To define a matrix, use the function matrix:

```
mat=matrix(data=c(9,2,3,4,5,6),ncol=3)
> mat
    [,1] [,2] [,3]
[1,]  9    3    5
[2,]  2    4    6
```

The argument data specifies which numbers should be in the matrix. Use either `ncol` to specify the number of columns or `nrow` to specify the number of rows.
Matrix-operations are similar to vector opera- tions:

```
> mat[1,2]
[1] 3
> mat[2,]
[1] 2 4 6
> mean(mat)
[1] 4.8333
```

- Elements of a matrix can be addressed in the usual way: `[row,column]` (first line).
- When you want to select a whole row, you leave the spot for the column number empty (third line).
- The last line shows that many functions also work with matrices as argument.

## 6.3 Data frames

Time series are often ordered in data frames. A data frame is a matrix with names above the columns. This is nice, because you can call and use one of the columns without knowing in which position it is.

```
> t = data.frame(x = c(11,12,14),
y = c(19,20,21), z = c(10,9,7))
> t
   x  y  z
1 11 19 10
2 12 20  9
3 14 21  7
> mean(t$z)
[1] 8.666667
> mean(t[["z"]])
[1] 8.666667
```

- In lines 1-2 a typical data frame called `t` is constructed. The columns have the names `x`, `y` and `z`
- Line 8-11 show two ways of how you can select the column called `z` from the data frame called `t`.

# 7 Graphics

Plotting is an important statistical activity. So it should not come as a surprise that `R` has many plotting facilities. The following lines show a simple plot:

```
> plot(rnorm(100), type="l", col="gold")
```

Hundred random numbers are plotted by connecting the points by lines (the symbol between quotes after the `type=`, is the letter `l`) in a gold color. Another very simple example is the classical statistical histogram plot, generated by the simple command

```
> hist(rnorm(100))
```

To learn more about formatting plots, search for `par` in the `R help`. Google "R color chart" for a pdf file with a wealth of color options. To copy your plot to a document, go to the plots window, click the "Export" button, choose the nicest width and height and click `Copy` or `Save`.

## 8 Reading and writing data files

There are many ways to write data from within the R environment to files, and to read data from files. We will illustrate one way here. The following lines illustrate the essential:

```
> d = data.frame(a = c(3,4,5),
b = c(12,43,54))
> d
  a  b
1 3 12
2 4 43
3 5 54
> write.table(d, file="tst0.txt",
row.names=FALSE)
> d2 = read.table(file="tst0.txt",
header=TRUE)
> d2
a b
1 3 12
2 4 43
3 5 54
```

- In lines 1-2, a simple example data frame is constructed and stored in the variable `d`.
- Lines 3-7 show the content of this data frame: two columns (called `a` and `b`), each containing three numbers.
- Line 8 writes this data frame to a text file, called `tst0.txt` The argument `row.names=FALSE` prevents that row names are written to the file. Because nothing is specified about `col.names`, the default option `col.names=TRUE` is chosen and column names are written to the file.
- Lines 10-11 illustrate how to read a file into a data frame. Note that the column names are also read. The data frame also appears in the workspace window.

## 9 An Introductory Exercise

The following data are the masses in grams of a random sample of size 50 from a cohort of young salmon. The data are assumed to come from a $N(\mu, \sigma^2)$ distribution, where $\mu$ and $\sigma$ are unknown.

**Salmon masses in grams**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 162 | 170 | 171 | 169 | 188 | 188 | 174 | 156 | 172 | 168 |
| 173 | 164 | 184 | 174 | 165 | 168 | 174 | 179 | 162 | 179 |
| 174 | 159 | 190 | 175 | 193 | 183 | 177 | 181 | 192 | 175 |
| 167 | 173 | 166 | 178 | 160 | 186 | 163 | 181 | 177 | 180 |
| 177 | 180 | 182 | 200 | 187 | 187 | 196 | 176 | 191 | 176 |

**1.** Construct a vector in R that contains the data.

**2.** Obtain basic summary statistics of the data, including the variance.

3. Use the functions `hist` and `boxplot` to plot the data. Additionally, a normal probability plot can be obtained using the function `qqnorm`.

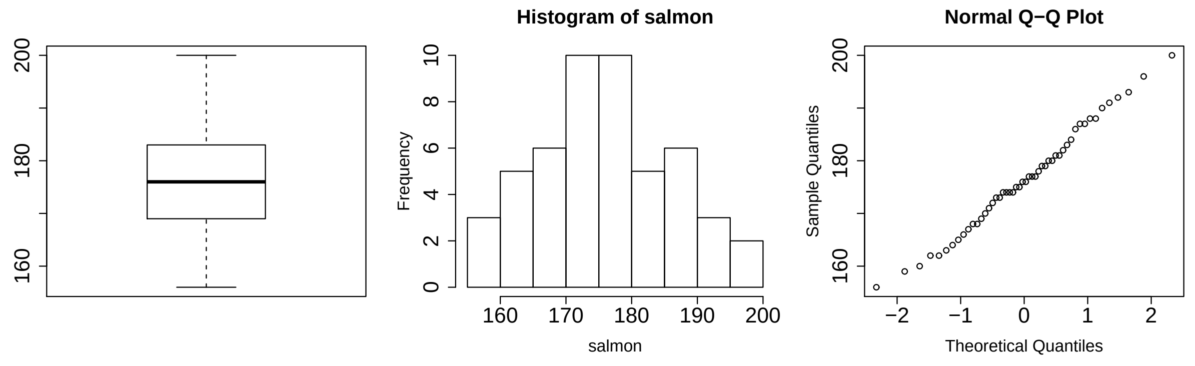
4. Obtain a 95% confidence interval for $\mu$.

5. Use R to calculate percentage points of the appropriate chi-square distribution and hence construct a vector that contains the end-points of a 95% confidence interval for $\sigma^2$.

The functions `qt(p,df)` and `qchisq(p,df)` can be used to obtain the relevant quantiles of the appropriate $t$ and $\chi^2$ distributions. The first argument `p` is the value of $1 - \alpha/2$ (where $\alpha$ is the chosen level of the test), and the second is the degrees of freedom, `df`.

A solution to this exercise is included overleaf. Don't worry if you need to look for help at this at this stage, but do be careful to see how the various elements of code relate to the questions. The graphs have been improved upon from the basic commands shown. There will be more about this later.

```
> salmon <- c(162,170, ... ,176)
> summary(salmon)
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  156.0   169.2   176.0   176.4   182.8   200.0
> v <- var(salmon)
>  v
[1] 103.2718
> boxplot(salmon)
> hist(salmon)
> qqnorm(salmon)
```



```
> #Confidence Interval for the mean
xbar <- mean(salmon)
> xbar
[1] 176.44
> SE <- sqrt(var(salmon)/length(salmon))
> SE
[1] 1.437163
> df <- length(salmon) - 1
> df
[1] 49
> tt <- qt(0.975, df)
> CI <- c(xbar - tt * SE, xbar + tt * SE)
> CI
[1] 173.5519 179.3281

> #Confidence Interval for the Variance
CL <- qchisq(0.025, df)
> CL
[1] 31.55492
> CU <- qchisq(0.975, df)
> CU
[1] 70.22241
> CI <- c((df * var(salmon))/CU, (df * var(salmon))/CL)
> CI
[1]  72.06132 160.36550
```

Note that the confidence interval for the mean could be obtained directly using the `t.test` function (the confidence level is 95% by default).

```
> t.test(salmon)       #equivalent to t.test(salmon,conf.level=0.95)

One Sample t-test

data:  salmon
t = 122.77, df = 49, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 173.5519 179.3281
sample estimates:
mean of x
   176.44
```

Here, the default test of $H_0 : \mu = 0$ is not very informative. Alternative tests, such as say $H_0 : \mu = 180$, can be undertaken using `t.test(salmon,mu=180)`.

Try this. Do you get the result you expected?

The function `t.test` can also be used to provide a two-sample $t$ test, *viz*

```
t.test(var1,var2,var.equal=T)
```

R also has an inbuilt function to test for equality of variances, using an $F$-test

```
var.test(var1,var2)
```

You will have an opportunity to investigate these functions further in the exercises provided.