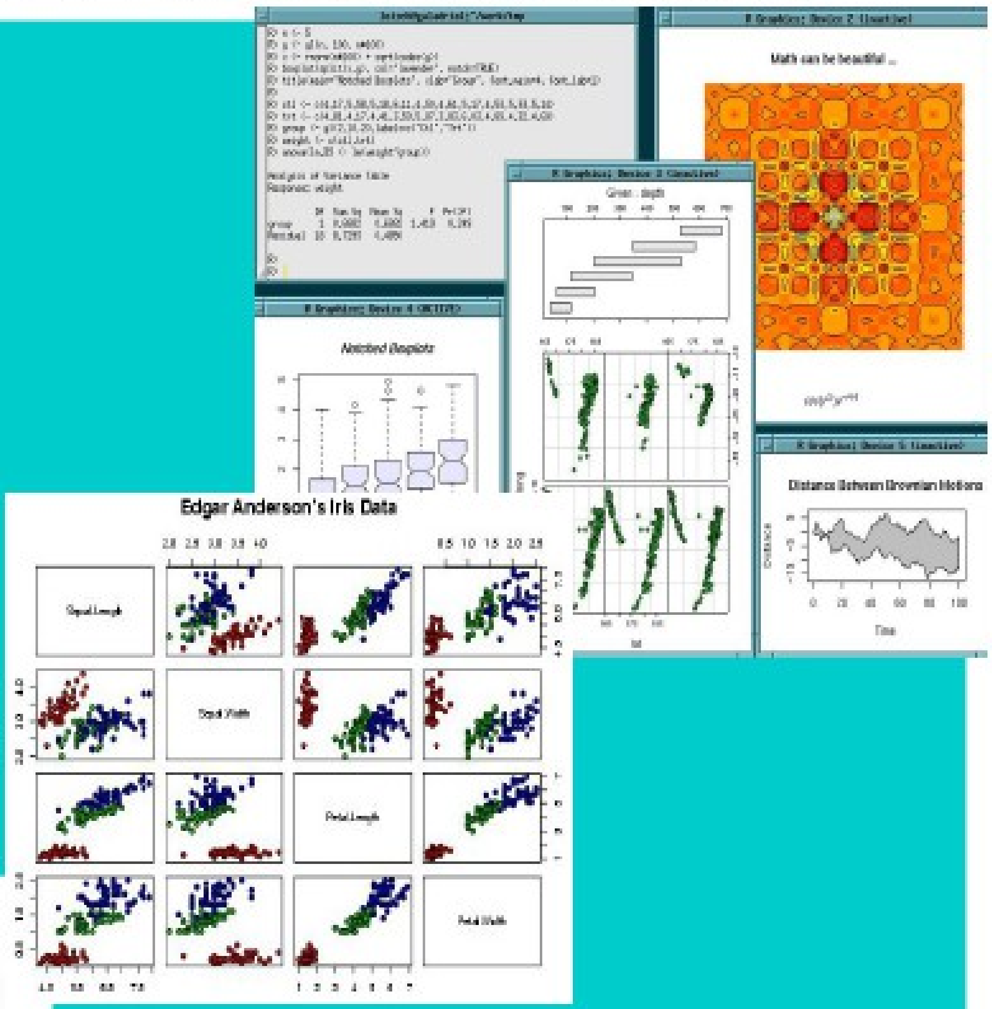
INTRODUCTORY STATISTICS AND PROGRAMMING WITH *R* FOR BIOLOGICAL SCIENCES


Math can be beautiful ...
Notched Boxplots
Edgar Anderson's Iris Data
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
Distance Between Brownian Motions

Module I: Fundamentals of R language

Aldo Barreiro Felpeto, June 2015

Table of contents

Preface

Why learning R?

1) Flexible and interactive language (“infinitely customizable to your problem”, no “black boxes”).
Example of being flexible and interactive: control of model structure in linear mixed models

```
#all covariates:
> model1<-lm(vardep~varind1+varind2+varind3+varind4+varind5)
#consider interactions of order 2 only:
> model2<-lm(vardep~varind1:varind2:varind3:varind4:varind5)
#consider interactions until order 3 only:
> model3<-lm(vardep~(varind1+varind2+varind3+varind4+varind5)^3)
#consider all possible interactions:
> model4<-lm(vardep~varind1*varind2*varind3*varind4*varind5)
#one random factor affecting the slope and the intercept:
> model5<-lme(vardep~varind1*varind2*varind3*varind4*varind5,random=~1|block1)
#one random factor affecting only the intercept:
> model6<-lme(vardep~varind1*varind2*varind3*varind4*varind5,random=~1|)
#nested random factors:
> model7<-lme(vardep~varind1*varind2*varind3*varind4*varind5,random=~1|block1/block2)
#all at once:
> model8<-lme(vardep~varind1*varind2:varind3+
(varind4+varind5+varind6+varind7)^3,random=~1|block1/block2)
```

Think about the difference in flexibility of R respect to other software (SPSS, STATISTICA) simply as the difference between an elaborated human language respect to any other language made of signs, for instance, traffic signs. The human language has elements structured at different levels (phonemes, words, phrases) and is highly articulated at each level. This can generate lots of different meanings with subtle changes. So, it can “customize” communication with high accuracy. A language like the traffic signs, with few levels and almost not articulated at all, can not account for subtle changes in the meaning of the messages.
But, …. the more elaborated languages are more difficult to learn as well. It is not worth learning a complicated language if you just want to communicate simple messages like: “you must stop”, “you are allowed to run as fast as 80 km per hour” etc. The same trade – off exists between learning to use R and other statistical software (SPSS, STATISTICA, etc.).
However, in their most recent versions, many other programs are becoming more interactive and flexible (in part, due to the success of R).
The “black boxes” present in other software are those processes of the analysis in which the user does not have control over one or several steps. So you might get an output without knowing exactly what the software has done, and you can not modify this process.
The absence of “black boxes” is, in some sense, a problem for certain R users. No “black boxes” policy implies that you have to know more about statistics in order to correctly use the software. Because you have to control more steps of the analysis and understand what is going on at each of them.

2) It is free.

3) Open source: lots of contributors, more applications than any other statistical software, quality of backup and help, credibility, no “black boxes”.
As an example of the importance of R: some of the most popular commercial statistical softwares make publicity of the features that connect their software to R applications. For instance, SPSS has many connections to R applications that make them available to be used with the SPSS interface.

4) Many applications already in packages (like S, SAS, SPSS,... but unlike C++).

5) It is relatively easy to incorporate routines from other languages in R: C, C++, Fortran (cross compilation).

6) No rivals in cutting-edge statistical applications (mixed models, generalized additive models, ...).

7) My favorite: “[…] many of the top people will have switched to R already. A large proportion of the world leading statisticians use R, and this should tell you something [...]” (M.J. Crawley, The R book, 2007).

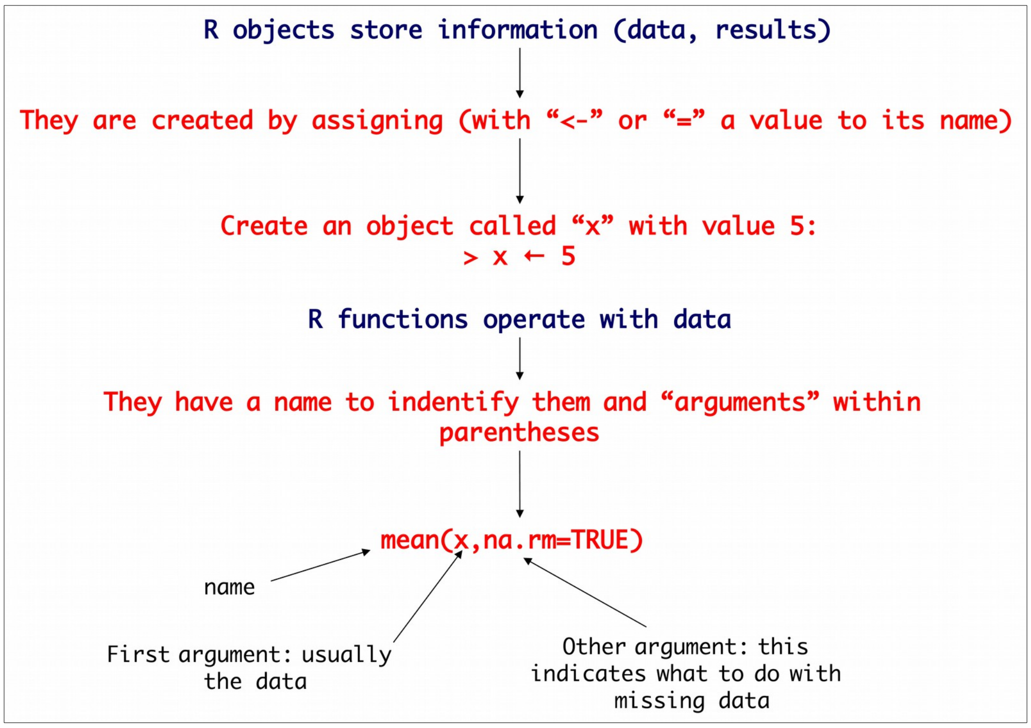
Many of the above mentioned advantages of using R are many different sides of the basic features of R. As I have pointed out a couple of times, the most common statistical softwares have improved some aspects in their recent versions in order to better compete with R.
It is important to keep in mind that some of the best features of R (such as the flexibility, the lack of black boxes) make this language less user-friendly than other statistical packages.
The point is not to do apology for the use of a specific software (either R or any other). The ultimate choice depends a lot on your needs regarding the depth of the statistical treatment of your data, your facilities regarding payment of software licenses, etc.
Be aware that, in order to be self-sufficient using R, you will need more time practicing than with other software designed to be more user-friendly (SPSS, STATISTICA, etc.). Although some GUIs (we will talk about it later) make the basic features of R almost as accessible as SPSS or STATISTICA (R Commander). The software is a choice to be made considering the trade – offs between time invested and reward.

Section 1: Basic of R language

Contents

1.1 First things to know about R

R is a programming language for statistics and mathematics created by Ihaka and Gentlemen in 1997. It was implemented based on the features of other two already existing languages: *S* and *Scheme*. Currently, its development and maintenance is performed by the R Development Core Team. R is part of the GNU project. Information and downloads can be found at http://www.r-project.org/
A few remarkable features of R:
- R is an interpreted language, which means that its functioning is due to the interpretation of text commands.
- R is an object – oriented language, which means that the information is stored in objects with different properties. The objects are created by assigning them a value.
- is polymorphic, which means that the same function can be applied to many different types of objects, and it works differently depending on the type of object and the argents passed to the function.
- R is vectorial on its operations. Each object is treated as a vector, or a set of vectors.
Below is a very simple schematic view of how the R language works:

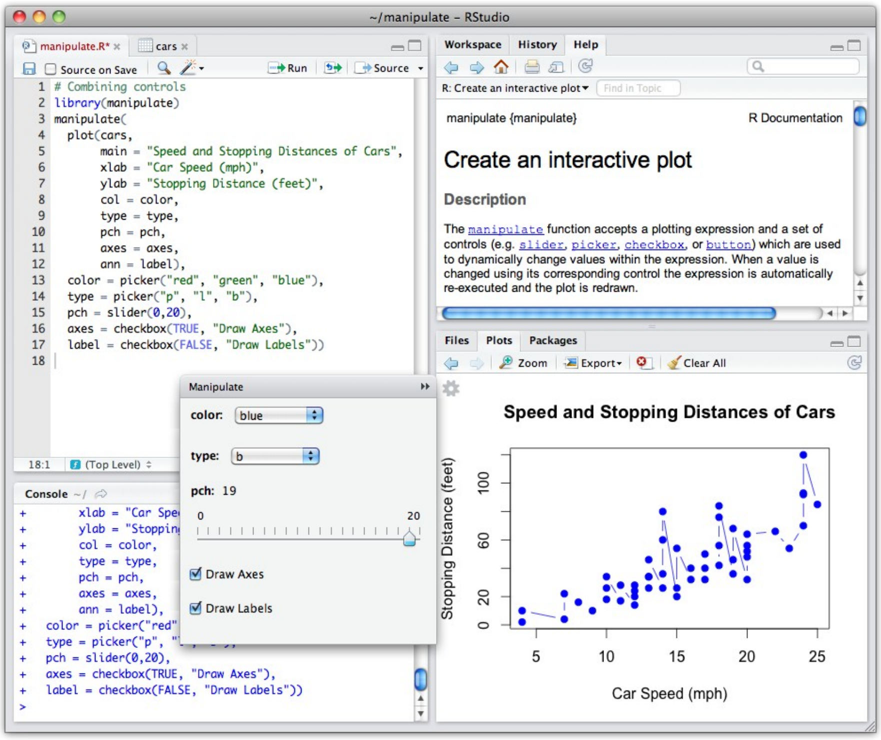

R objects store information (data, results)
They are created by assigning (with "<-" or "=" a value to its name)
Create an object called "x" with value 5:
> x ← 5
R functions operate with data
They have a name to indentify them and "arguments" within parentheses
mean(x,na.rm=TRUE)
name
First argument: usually the data
Other argument: this indicates what to do with missing data

Starting with R (file Rcourse.1)

I assume that each student got the necessary instructions for downloading and installing R by e-mail prior to the start of this course. Then, we go ahead to open an R session. We open a new R session by clicking in the program icon or by typing "r" in the Terminal.

There are many possible graphical User interfaces (GUIs) for R, depending also on the operating system. The basic installation, irrespective of the operative system, provides a very simple GUI, with a few displaying menus. There are other available GUIs that

incorporate more features, like R Commander, Tinn-R, RKward (very similar to S-PLUS). They are not available for all operating systems.
There is also an IDE (integrated development Environment) for R, called RStudio. It is a graphical users interface as well, but that makes more accessible all the “dispersed” features of R. Here are a picture of how the RStudio and R Commander interfaces:



```
# Combining controls
library(manipulate)
manipulate(
  plot(cars,
       main = "Speed and Stopping Distances of Cars",
       xlab = "Car Speed (mph)",
       ylab = "Stopping Distance (feet)",
       col = color,
       type = type,
       pch = pch,
       axes = axes,
       ann = label),
  color = picker("red", "green", "blue"),
  type = picker("p", "l", "b"),
  pch = slider(0,20),
  axes = checkbox(TRUE, "Draw Axes"),
  label = checkbox(FALSE, "Draw Labels"))
```

manipulate {manipulate} R Documentation

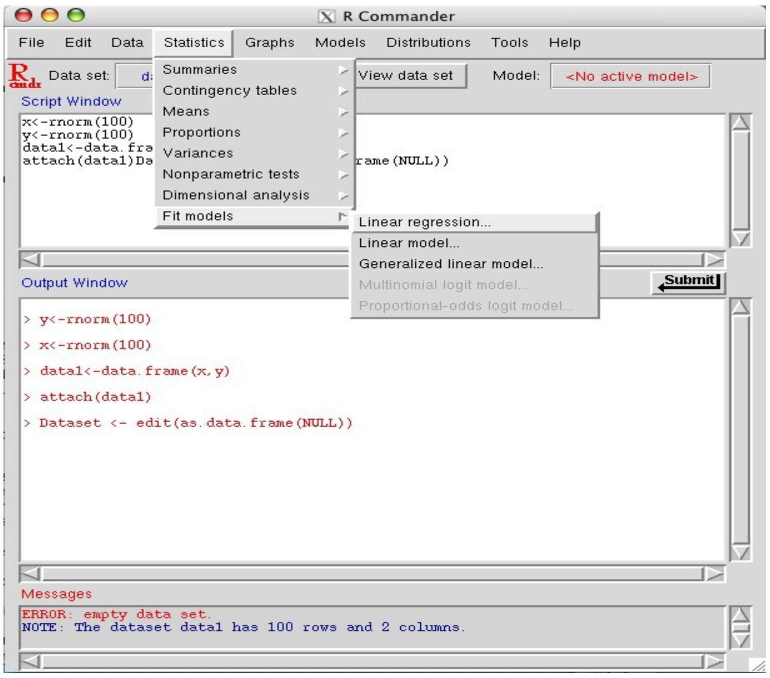
Create an interactive plot

Description

The manipulate function accepts a plotting expression and a set of controls (e.g. slider, picker, checkbox, or button) which are used to dynamically change values within the expression. When a value is changed using its corresponding control the expression is automatically re-executed and the plot is redrawn.

```
+        xlab = "Car Spe
+        ylab = "Stoppin
+        col = color,
+        type = type,
+        pch = pch,
+        axes = axes,
+        ann = label),
+   color = picker("red"
+   type = picker("p", "
+   pch = slider(0,20),
+   axes = checkbox(TRUE, "Draw Axes"),
+   label = checkbox(FALSE, "Draw Labels"))
>
```

color: blue
type: b
pch: 19
☑ Draw Axes
☑ Draw Labels



```
x<-rnorm(100)
y<-rnorm(100)
data1<-data.fra
attach(data1)Da
```

```
> y<-rnorm(100)
> x<-rnorm(100)
> data1<-data.frame(x,y)
> attach(data1)
> Dataset <- edit(as.data.frame(NULL))
```

```
ERROR: empty data set.
NOTE: The dataset data1 has 100 rows and 2 columns.
```

An important thing to do at the beginning of an R session is setting the working directory. From this directory we will import the files we need, with data and/or code, and

export the files with results and/or data that we generate in R. Usually, the GUIs have a menu from which you can select graphically the working directory. In any case, there is a *built-in* function in R to set the working directory, which is setwd():
MacOS notation:

```
> setwd("/Users/aldobarreirofelpeto")
```

The windows notation is a bit different:

```
> setwd("C:/Users/aldobarreirofelpeto")
```

In you want to know your actual working directory, use the function getwd():

```
> getwd()
[1] "/Users/aldobarreirofelpeto"
```

Now, we are going to start using examples from scripts. A script is a plain text file where we can write and save our commands for future work sessions. We will see later how to create a simple script structuring the commands to perform a simple analysis.
In order to execute the commands from a script, there are several options. You can just place the cursor on the command line to execute and execute it by typing *Ctrl* + *R* in the keyboard. If you want to execute at once more than one command, select the whole commands and type *Ctrl* + *R*. In Mac there is no *Ctrl* + *R* option, you have to execute from a menu in the GUI. For this course, however, I recommend to, mostly, write the commands on your R console. Avoid copying and pasting commands, since if you get used to this, it will cause errors, particularly when copy-pasting from pdf files.
How to stop a currently executing command is also a good thing to know. In Windows and Mac, you can do it by typing “Esc”. In Mac there is a menu in the GUI as well.

* It is possible to run simultaneous R sessions in your computer, in the same desktop. It depends on your operating system how to proceed. In MacOS you can run one session by clicking the R application icon and starting other one in the bash. Another option for MacOS is to duplicate the application. You have to go to your Applications folder and secondary-click the R icon, then select “duplicate” and a new copy of the application appears, which you could run simultaneously with a session opened from the original R application icon. In PC is very simple to start new sessions by clicking in the main icon, there is no need to duplicate the application.

As we mentioned before, the applications in R are distributed in packages. These applications are groups of functions coded to perform specific tasks. When you download R, you get installed a series of packages (*base*, *utils*, *stats*, *graphics*, *datasets*, *MASS*,...) containing *built-in* functions of common use, as well as some data sets. The majority of these packages are loaded by default in your R session as soon as you open it (*base*, *utils*, *datasets*, *stats*, *graphics*) but others are not loaded, and you have to load them if you need their functions (*MASS*). If you need applications from packages that are not installed, you will need to install them from the R repositories, and load them in your session. We will learn these processes while performing our exercises.
As we said above, R is an object – oriented language, so all the information that we generate is stored in the memory of an object. Let's start creating a very simple object:

We will call this object “x” and it is going to store the value of the integer “5”

```
> x<-5
```

The value of the object is “assigned”. The symbol for assigning is the “<-” but it could be used the symbol “=” as well (so “x<-5” is equivalent to “x=5”).
Now, if we type the name of the object and “enter”, R returns the content of the object:

```
> x
[1] 5
```

Remember what we said somewhere above: objects in R are considered vectors. So, our “x” is a vector of dimension 1, with length = 1. The number 1 inside the brackets indicates that the element next to it is the first in the sequence of components of the object “x”.

Let's create a vector of greater length. Now, a one dimensional vector with length = 6 instead of 1.

```
> u<-c(5,9,13,1,3,9)
```

If we “ask” R about our new object “u”, it will tell us:

```
> u
[1]  5  9 13  1  3  9
```

*An important feature of R language is that it is case sensitive, which means that it differentiates lower from upper case. So our object is called “x”, not “X”:

```
> X
Error: objeto 'X' no encontrado
```

* The sybol “#” is to separte a coment from the actual code. Everything to the right of a “#” will not be read by R.

Right:

```
> x #Our first example
[1] 5
```

Wrong:

```
> x Our first example
Error: inesperado símbolo en "x Our"
```

Now it is time to talk a bit about a key component of R that we have already used: the *built-in* functions. The *built-in* functions are coded in the packages that you have installed and loaded in the session. These functions are used typing their names and parentheses, which contain the function arguments. These arguments are the objects over which the functions operate as well as other kind of specifications that modulate the action of the function.

In order to create “x” and “u”, we used a very simple *built-in* function: c(). “c” stands for “concatenate”. What this function does is simply concatenating elements to create a vector of dimension = 1. In our case, we called this vector “u”.
Let's create objects with greater dimensions, for instance, a 3x3 matrix. As usually in R, there are different ways to perform a specific task. For example, we can use the function matrix(), indicating which elements are to be contained in the matrix, and the dimensions of the matrix:

```
> A<-matrix(data=c(5,9,13,1,3,9,12,2,17),nrow=3,ncol=3,byrow=T)
> A
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    1    3    9
[3,]   12    2   17
```

As we can see, the first argument in the function matrix() is a vector with the matrix components. We also specified the number of rows with the argument “nrow” and the number of columns with the argument “ncol”. The argument “byrow” specifies in which direction to read the numbers in order to construct the matrix. By default, this is performed column-wise. So, in this example, we modified the default option.
Remember again that R considers every object as a vector, which is particularly important when performing operations. So, for R, our matrix “A” is a vector that has three sub-vectors (each column a vector, this is why by default a matrix is read column-wise).
Another way of creating a matrix is adding dimension 2 to an already existing one – dimensional vector. For instance, we want to transform our vector “u” in a 3x2 matrix:

```
> dim(u)<-c(3,2) #R always reads first the rows, then the columns
> u
     [,1] [,2]
[1,]    5    1
[2,]    9    3
[3,]   13    9
```

Now, let's introduce another key feature from R. The HTML help. There are many functions in R, with many different arguments, in different packages. There is no way you can remember all of them. So the help will show you how to use a function (which arguments it has, the possible values of the arguments, what the function returns when applying it, and coded examples of the usage of the function). The function for the HTML help is help(), and it can we abbreviated as “?”
An example:

```
> help(mean)
> ?mean
```

* Now that we know about the help, a few comments about functions arguments: if we write the arguments in the function in the correct position (the position shown in the HTML help page) we do not have to write the argument name, only its value. If we do not want to change the default value of an argument, we do not need to write this argument. As an example of these things, let's create another matrix:

```
> ?matrix
> B<-matrix(c(5,10,23,4,5,2),3,2)
```

So this matrix will have the elements of the first vector, with 3 rows and 2 columns, because we wrote all these three argument values in the correct place and the elements will be inserted column-wise (because we did not specified anything in the argument “byrow”, its default value is applied, which is “TRUE”). And the rows and columns will not have names, because we did not use the argument “dimnames”:

```
> B

     [,1] [,2]
[1,]    5    4
[2,]   10    5
[3,]   23    2
```

It would be the same result like this:

```
> B2<-matrix(ncol=2,data=c(5,10,23,4,5,2),nrow=3)
> B2
     [,1] [,2]
[1,]    5    4
[2,]   10    5
[3,]   23    2
```

But the HTML help only applies to the functions contained in the packages that you have INSTALLED AND LOADED in your session. And you have to type the EXACT function name, otherwise it would not work:

```
> help(Mean)
No documentation for ‘Mean’ in specified packages and libraries:
you could try ‘??Mean’
```

If you are not sure about the exact function name, there is another function that performs a search within ALL THE INSTALLED packages, not just the loaded ones. And will return the name of the functions found with a name similar to the one you are looking for, as well as the package were they are available. This function is help.search(), which performs its search in the help system with fuzzy matching (that is why you do not have to type the exact name of the function). It can be abbreviated as ??

```
> help.search(“linear model”) #Remember: “item to search”
> ??”linear model”
```

The help systems are very important, you will use the help almost every R session. There is people who learned to program with R by themselves, just knowing how to use the help features.

A sometimes useful command is the one that removes all the objects created in a session:

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

Closing an R session is as simple as typing q() in the console or clicking the “close” button. You will be asked about saving the current workspace. By doing it, you will save the objects stored in the memory of the session.

As a quick start with the very basic features of R, this was enough. What you have learned so far are the foundations of what you should know in order to use R. Every new features that we will learn so forth, will come like branches from this stem.

1.2 Objects

Above, we have explained what an object is and its main purpose, which is storing data or results from analysis. We have created, in a very simple and straightforward way, a couple two elemental kinds of objects: vectors and matrices. A vector is the most simple kind of object. It has one only dimension, which in R is equivalent to a matrix column, and at least one element, which position is equivalent to row numbers in a matrix. A matrix is a two-dimensional object, with dimension 1 being rows and dimension 2 being columns. When operating with matrices, columns are considered independent vectors.
But there are also other important kinds of objects:
- Arrays: 3 dimensions, they could be created with the array() function.
- Lists: sets of components of, potentially, different classes (we will see below what “class” is). They could be created with the list() function.
It is very common that the results of a statistical test, performed with a single function are returned as a list. Because these results might be a set of objects of different classes.
- Data frames: groups of data similar to matrices, but by default they consider that variables are set by columns, and by default include variable names as column names. They could be created with the data.frame() function. These are the ones we will use more when working with data matrices. Usually, when be import a data set into R from a .xls, .txt or similar file, it is imported by default as a data.frame.
R packages contain several data frames that are data sets, they are very useful for learning statistical techniques and working with R functions. Some packages provide data sets that are specific to try the functions coded in that package. There is a package, *datasets*, that has only data sets, like: *iris*, *airquality*, *CO2*, *cars*, … check them in the HTML help of this package!

```
> data(airquality)
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
```

```
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

The “class” is an attribute of the R objects. It can be defined by a vector of characters stating the name of the class:
- Numeric.
- Integer.
- Logical.
- Character.
It is important to mention that the class character is specified with “ ”. So that if we want to create a vector of characters:

```
> ch<-c("a","b","A","B")
> ch
[1] "a" "b" "A" "B"
```

Or, it could be an implicit attribute of certain objects like:
- Matrix
- Array
You can ask which is the class of an object with the class() function:

```
> class(u)
[1] "matrix"
> class(ch)
[1] "character"
```

Objects also have a mode, which is the form that the object has in the program memory.

```
> test<-TRUE
> mode(test)
[1] "logical"
> mode(u)
[1] "numeric"
> mode(ch)
[1] "character"
```

There is another important attribute of objects, particularly in the case of data frames, which is the structure. Let's ask R about the structure of a famous data frame that is provided in the *datasets* package, the data frame *iris*:
In order to load this data frame in your session, you need the function data():

```
> data(iris)
```

If we want to know something about the origin and features of these data:

```
> ?iris
```

And now ask about its structure, with the function str():

```
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Let's see an example showing how lists are frequently used. We are going to perform a Student's test, *t-test*, comparing the sepal length of two of the species, and store the results in an object called "results":

```
> results<-
t.test(iris[iris$Species=="versicolor","Sepal.Length"],iris[iris$Species=="setosa","Sepal
.Length"]) #Forget about the indexing so far!!!
> results
       Welch Two Sample t-test

data:  iris[iris$Species == "versicolor", "Sepal.Length"] and iris[iris$Species ==
"setosa", "Sepal.Length"]
t = 10.521, df = 86.538, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.7542926 1.1057074
sample estimates:
mean of x mean of y
    5.936     5.006

> class(results)
[1] "htest"
> mode(results)
[1] "list"
```

1.3. Operators

There are many operators... arithmetic: + , -, *, /, ^ (sign for power), logic: & ("and"), | ("or"), ! ("no" or "different from"), == ("equal" or "the same as"), others: NA ("not available", usually a missing data), NaN ("not a number" returned from an operation that has no result), -Inf, Inf, etc. matrix operations: %*% (matrix multiplication, be careful: * returns element-wise multiplication), %o% (outer product).
You can check them at the HTML help:

```
> ?Arith
> ?Compare
```

1.4. Functions

Common built-in functions from the default downloaded and installed packages:
- Arithmetic: log(), log10(), exp(),sqrt(),...
- Trigonometric: sin(), cos(), tan(),...

- Complex: Arg(), Conj(), Im(),...
- Conversion between object attributes: as.numeric(), as.character(),...
- Properties of objects: is.numeric(), is.character(),...
- Summaries: max(), min(), range(), sum(),...
- Statistic: mean(), var(), sd(), median(), lm(), aov(), rnorm(),...
- Graphics: plot(), hist(), boxplot(), pie(),...
- Loops: apply(), sapply(), lapply, mapply(), tapply()...

You can check them in the HTML help:

```
> ?Math
> ?Math2
> ?Summary
> ?Complex
```

Using common built-in functions **(file Rcourse.2)**

Now we are going to have a look at examples of the use of relevant *built-in* functions that are present in the default installed and loaded packages, grouping them into categories based on the kind of tasks that they perform. We are going to use our examples of functions with some objects created ad-hoc, as well as data sets from the *datasets* package.

```
> data(airquality)
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

Summary functions

They are useful for calculating summary values from data.
Calculate a maximum

```
> max(airquality$Temp)
[1] 97
```

Calculate a minimum

```
> min(airquality$Temp)
[1] 56
```

Calculate a range

```
> range(airquality$Temp)
[1] 56 97
```

Calculate a cumulative sum

```
> sum(airquality$Solar.R)
[1] NA
```

This one does not work if there are NA in the vector!
Check the help:

```
> ?sum
```

So, we see that we have to change the default value of the argument “na.rm”:

```
> sum(airquality$Solar.R,na.rm=TRUE)
[1] 27146
```

Statistical functions

These are self – explanatory:

```
> mean(airquality$Ozone,na.rm=T)
[1] 42.12931
> median(airquality$Ozone,na.rm=T)
[1] 31.5
> sd(airquality$Ozone,na.rm=T)
[1] 32.98788
> var(airquality$Ozone,na.rm=T)
[1] 1088.201
```

There is a family for generating random numbers, withdrawn from specific distributions:

```
> rnorm(10)
 [1] -1.59191752  0.40536897 -0.90492153 -1.05683395 -1.04285024  1.14036009 -0.25691899
 [8]  2.18496568 -0.24035442 -0.09505727
> rnorm(15,mean=mean(airquality$Ozone,na.rm=T),sd=sd(airquality$Ozone,na.rm=T))
 [1]  30.848315   7.876733  37.930143 -18.509056  81.759604  49.036042  44.889502
72.786658
 [9]  13.941220  55.094098  45.806069  20.704722  33.920354  19.854788  10.767707
> ?rnorm
```

There are several functions for performing simple statistical tests:

- Anova

```
> data(airquality)
> model1<-aov(airquality$Ozone~airquality$Solar.R+airquality$Temp)
> summary(model1)
                    Df Sum Sq Mean Sq F value   Pr(>F)    
airquality$Solar.R   1  14780   14780   26.76 1.07e-06 ***
airquality$Temp      1  47378   47378   85.79 2.22e-15 ***
Residuals          108  59644     552                     
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
42 observations deleted due to missingness
```

- Student's t-test

```
t.test(airquality[airquality$Month==5,"Temp"],airquality[airquality$Month==8,"Temp"])

     Welch Two Sample t-test

data:  airquality[airquality$Month == 5, "Temp"] and airquality[airquality$Month == 8,
"Temp"]
t = -10.789, df = 59.904, p-value = 1.138e-15
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -21.83446 -15.00425
sample estimates:
mean of x mean of y
 65.54839  83.96774
```

Non parametric tests:

- Wilcoxon test for 2 groups (equivalent to Mann – Whitney test)

```
> wilcox.test(Ozone~Month,data=airquality,subset=Month%in%c(5,8))

     Wilcoxon rank sum test with continuity correction

data:  Ozone by Month
W = 127.5, p-value = 0.0001208
alternative hypothesis: true location shift is not equal to 0

Mensajes de aviso perdidos
In wilcox.test.default(x = c(41L, 36L, 12L, 18L, 28L, 23L, 19L,  :
  cannot compute exact p-value with ties
```

Note how useful is the argument “data” (common to many other functions). With it, you do not have to write the “$” to locate a variable inside the data frame. There is a similar alternative to do this with the function attach(), but I do not recommend using it, it is dangerous when you are working with variables with the same name in different data frames.

- Kruskal – Wallis anova for *k* groups:

```
> kruskal.test(Ozone~Month,data=airquality)

     Kruskal-Wallis rank sum test

data:  Ozone by Month
Kruskal-Wallis chi-squared = 29.2666, df = 4, p-value = 6.901e-06
```

* You can see the code of a function by simply typing the function name without parentheses and click enter:

```
> sd
function (x, na.rm = FALSE)
{
    if (is.matrix(x)) {
        msg <- "sd(<matrix>) is deprecated.\n Use apply(*, 2, sd) instead."
        warning(paste(msg, collapse = ""), call. = FALSE, domain = NA)
        apply(x, 2, sd, na.rm = na.rm)
    }
    else if (is.vector(x))
        sqrt(var(x, na.rm = na.rm))
    else if (is.data.frame(x)) {
        msg <- "sd(<data.frame>) is deprecated.\n Use sapply(*, sd) instead."
        warning(paste(msg, collapse = ""), call. = FALSE, domain = NA)
        sapply(x, sd, na.rm = na.rm)
    }
    else sqrt(var(as.vector(x), na.rm = na.rm))
}
<bytecode: 0x1bb0f48>
<environment: namespace:stats>
```

Text

There are functions for writing text in R. They are not going to be very useful in this course, with the exception of those for writing text in plots, that we will see in the next section. So we are not going through examples, but at least mention some: cat() concatenates characters, with several options to bind them. print() prints its argument. paste() transforms objects into strings of characters and then concatenates them in a single string. There are several functions for processing text: substr() gives access to characters inside a string, strsplit() splits strings.

User-defined functions **(file Rcourse.2)**

When working with R it is very important to define your own functions. We are going to see a simple example. In order to define a function, you have to assign to its name a structure that defines its arguments within parentheses and its code within keys. Often, the function return() is used in order to specify the ultimate value to return from the function.

```
> myfunct<-function(x,a){
+     tot<-sum(x)
+     pow<-tot^a
+     return(pow)
+ }
>
> z<-c(34,56,78,90)
> myfunct(z,3)
```

```
[1] 17173512
```

Generate output files **(file Rcourse.2)**

It is very common to generate simple output files storing the results of the commands that are executed in a session, instead of showing them in the screen. This is useful for the storage of results from statistical tests. Here is a simple example of how ths could be done, with the function sink():

```
> out<-"Myfile.txt"
> sink(out)
> data(airquality)
> head(airquality)
> t.test(airquality[airquality$Month==5,"Temp"],airquality[airquality$Month==8,"Temp"])
> sink()
```

First, we created an object storing the file name and extension (in this case, we called it "out"). Then, we "open" this output with the function sink(), indicating within the parentheses which is the output file that will gather the results. Then, we write the commands fo the operations that we aim to do, and finally, we siply close the output file with sink(). It is very important to remember to close the output files.

1.5 Packages

* Another feature of the help is the information about packages. You have all of the information you need in the pdf manual in the R project web. But if you want some information about a specific package to appear in your R session, you can type:

```
> library(help=datasets)
```

Install and load packages

In the GUIs, there is usually a menu to perform this task. But you can also use functions with commands. For package installation:

```
> install.packages("name of the package", dep=TRUE)
```

Then, a list with the world R repositories appears, the "CRAN mirrors". You should select the closest one in geographical terms. If you are in Europe, any european mirror should work. If you "suspect" about the quality of the mirrors, that one that you should always trust is the one from Austria, since it is the central one of the R core developers.
It is important that you specify the argument "dep=TRUE". This is because there are packages that need subsidiary functions from other packages, and if you do not have them installed, the functions from this package may not work properly. With the argument "dep=TRUE", you will install all those packages that contain the subsidiary functions. Those packages are called "dependencies".
If you install the packages from the GUI menus, after selecting a mirror, a list with all the R

packages will appear, and then you have to select the one you want to install. Somewhere in the menus or dialog boxes should appear an option for installing the dependencies.
You can load a package as with menus from the GUI as well. Otherwise, the functions to perform it in the R console, with commands are:

```
> library(name of the package)
```

or

```
> require(name of the package)
```

Update R and packages

In order to update R to a newer version, without losing your installed packages, you have to do three things: first, you have to uninstall the old R version. Second, you have to go to the system folder containing you current R version in your operating system, and change the folder name to the name of the new R version (important to type the name correctly). Third, you have to download the new R version. It is important to say “YES” when asking if you want to install R in the already existing folder.
For the update of packages, you can usually go through a series of menus in the GUIs. But if you prefer to use functions in the R console you have to write this command:

```
> update.packages()
```

Then, R will show you one by one the available versions of each package, and if you want to update, you just type “yes”.
If you want to update a specific package, use the above function this way:

```
> update.packages(type=”name of the package”)
```

Demos

Many packages contain demos that show you briefly what kind of things could be done with that packages. The function is very simple:

```
> demo(graphics)
```

Then, a series of plots will appear (you usually have to click “enter” in order to change page), together with the scripts containing the code that was used to generate those plots.
There is also another useful function to show the utilities of specific functions:

```
> example(barplot)
```

Section 2: Operating with objects

Contents

2.1 Operating and indexing with vectors (file Rcourse.3)

The process of locating and selecting data is a very important one in programming or performing statistical analysis. Here we will show how can be performed with R. Selecting elements in vectors of dimension = 1. Let's create a new vector for this purpose, with the function: seq()

```
> w<-seq(2,16,2)
```

The simplest way to locate an element is by typing the ordinal number of that element within brackets. So, if we would like to select the third element of "w":

```
> w[3]
[1] 6
```

If we want to choose more than one element, we need the concatenate function, c():

```
> w[c(3,5)]
[1]  6 10
```

The function which() is very useful in order to set logical conditions for selecting elements:

```
> which(w>=10)
[1] 5 6 7 8
```

This returns the position of elements of "w" that agrees with the condition. But:

```
> w[which(w>=10)]
[1] 10 12 14 16
```

If we write the function inside the brackets, it will return the value of the elements that occupy those positions.
The function any() returns a logical with respect to the specified condition:

```
> any(w<10)
[1] TRUE
> any(w>20)
[1] FALSE
```

We might we interested in deleting elements from a vector:

```
> w[-4]
[1]  2  4  6 10 12 14 16
> w[-c(4,5)]
[1]  2  4  6 12 14 16
> w[-which(w>=10)]
[1] 2 4 6 8
```

2.2 Operating and indexing with matrices and data frames (file Rcourse.3)

Indexing in matrices, data frames and arrays is very similar. The only thing to take into account is that there are more dimensions. Each dimension inside the brackets is separated by a comma “,”. The position to the left of the “,” indicates rows, the position in the right indicates columns, and in the case of the arrays, there is a second “,” separating the third dimension. For example, we are going to select the element in the third row and second column from the data frame “dat”. For this purpose, in this example, we have to import a data set that we have in .csv file. Later, we will show in detail how to import data into an R session, now just worry about the indexing:

```
> dat<-read.csv2("Growth.csv",sep=",")
> head(dat)
    NO2         r
1 48.81 0.8154953
2 48.82 0.8154953
3 48.83 0.9640916
4 97.40 0.9640916
5 49.65 0.6424458
6 49.75 0.6424458
> dat[3,2]
[1] 0.9640916
```

If we need all the elements from a column, the place for the rows (left to the comma) should be empty. This indicates “I want all the rows”:

```
> dat[,2]
 [1] 0.815495325 0.815495325 0.964091604 0.964091604 0.642445838 0.642445838 0.486470530
 [8] 0.486470530 0.419487008 0.419487008 0.288947095 0.288947095 0.416609389 0.640224779
[15] 0.629875001 0.732448969 0.037589208 0.001508723 0.113558910
```

Correspondingly, if we wanted the elements from all the columns:

```
> dat[3,]
    NO2         r
3 48.83 0.9640916
```

We can complicate things a bit more, and introduce how the “:” works in order to create a sequence of integers with step size = 1:

```
> dat[4:10,2]
[1] 0.9640916 0.6424458 0.6424458 0.4864705 0.4864705 0.4194870 0.4194870
```

If the rows or columns had names, it is also possible to select by name. But name is a character, so it has the signs “ ”:

```
> dat[,"NO2"]
 [1]  48.81000  48.82000  48.83000  97.40000  49.65000  49.75000  49.67000  49.72000
 [9]  21.67000  22.00522   0.90000   0.80000 111.21611 340.00000 260.00000 100.00000
[17]   0.50000   0.40000   0.50000
```

We can try a few more complicated ways of selecting data with a more complex data frame, the data set *iris*:

```
> data(iris)
> iris[(iris$Sepal.Length==5)&(iris$Species=="setosa"),1:3]
   Sepal.Length Sepal.Width Petal.Length
5             5         3.6          1.4
8             5         3.4          1.5
26            5         3.0          1.6
27            5         3.4          1.6
36            5         3.2          1.2
41            5         3.5          1.3
44            5         3.5          1.6
50            5         3.3          1.4

> iris[iris$Sepal.Width>mean(iris$Sepal.Width),c("Sepal.Width","Species")]
    Sepal.Width    Species
1           3.5     setosa
3           3.2     setosa
4           3.1     setosa
5           3.6     setosa
6           3.9     setosa
7           3.4     setosa
8           3.4     setosa
10          3.1     setosa
11          3.7     setosa
12          3.4     setosa
15          4.0     setosa
16          4.4     setosa
17          3.9     setosa
18          3.5     setosa
19          3.8     setosa
20          3.8     setosa
21          3.4     setosa
22          3.7     setosa
23          3.6     setosa
24          3.3     setosa
25          3.4     setosa
27          3.4     setosa
28          3.5     setosa
29          3.4     setosa
30          3.2     setosa
31          3.1     setosa
32          3.4     setosa
33          4.1     setosa
34          4.2     setosa
35          3.1     setosa
36          3.2     setosa
37          3.5     setosa
38          3.6     setosa
40          3.4     setosa
41          3.5     setosa
43          3.2     setosa
44          3.5     setosa
```

```
45          3.8     setosa
47          3.8     setosa
48          3.2     setosa
49          3.7     setosa
50          3.3     setosa
51          3.2 versicolor
52          3.2 versicolor
53          3.1 versicolor
57          3.3 versicolor
66          3.1 versicolor
71          3.2 versicolor
86          3.4 versicolor
87          3.1 versicolor
101         3.3  virginica
110         3.6  virginica
111         3.2  virginica
116         3.2  virginica
118         3.8  virginica
121         3.2  virginica
125         3.3  virginica
126         3.2  virginica
132         3.8  virginica
137         3.4  virginica
138         3.1  virginica
140         3.1  virginica
141         3.1  virginica
142         3.1  virginica
144         3.2  virginica
145         3.3  virginica
149         3.4  virginica
```

There is also a very convenient function that could be used to select observations in data sets structured with factors, the function subset():

```
> subset(iris,Sepal.Length>=5&Sepal.Width>=4,select=c(Sepal.Length,Sepal.Width,Species))
   Sepal.Length Sepal.Width Species
15          5.8         4.0  setosa
16          5.7         4.4  setosa
33          5.2         4.1  setosa
34          5.5         4.2  setosa
```

Notice, from most of the examples above, how important is to use correctly the symbols “ ” and $.

2.3 Operating and indexing with lists (file Rcourse.3)

Basically, this is the way to proceed with matrices, data frames and arrays (with these, it would be the same, just adding a second comma to separate the third dimension). Now we have completed the most important indexing issues relative to the most common objects in R. Now let's see a few examples with lists, for which there are some subtle differences.
First, let's create a simple list:

```
> exV<-c(letters[1:10]) #We introduce the built-in object "letters"
> exV
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
> exM<-matrix(rep(5,25),5,5) #We introduce the function rep()
> exM
     [,1] [,2] [,3] [,4] [,5]
[1,]    5    5    5    5    5
[2,]    5    5    5    5    5
[3,]    5    5    5    5    5
[4,]    5    5    5    5    5
[5,]    5    5    5    5    5
> ex<-"Results"
> ex
[1] "Results"
> mylist<-list(exV,exM,ex)
> mylist
[[1]]
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

[[2]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    5    5    5    5    5
[2,]    5    5    5    5    5
[3,]    5    5    5    5    5
[4,]    5    5    5    5    5
[5,]    5    5    5    5    5

[[3]]
[1] "Results"
```

First thing about indexing in lists, the brackets ([ ]) are now double brackets ([[ ]]).

```
> mylist[[2]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    5    5    5    5    5
[2,]    5    5    5    5    5
[3,]    5    5    5    5    5
[4,]    5    5    5    5    5
[5,]    5    5    5    5    5
```

And we can also select sub-elements within a list (combining double and single brackets):

```
> mylist[[1]][3]
[1] "c"
```

Section 3: Functions of the “apply” family and loops

Contents

3.1. Loops (file Rcourse.4)

In order to built complex programs, we need to implement control structures, like loops. There are different ways to perform loops in R. The most common is with *for*, but you could implement a loop with the functions *while* and *repeat*. They are not used in the same way as a built-in function. You need to specify within brackets the operation to be performed. Here are is a simple example with *for*:

```
> res<-NULL
> for (i in 1:nrow(airquality)){res[i]<-airquality$Ozone[i]^3}
> res
  [1]   68921   46656    1728    5832      NA   21952   12167    6859     512      NA
 [11]     343    4096    1331    2744    5832    2744   39304     216   27000    1331
 [21]       1    1331      64   32768      NA      NA      NA   12167   91125 1520875
 [31]   50653      NA      NA      NA      NA      NA      NA   24389      NA  357911
 [41]   59319      NA      NA   12167      NA      NA    9261   50653    8000    1728
 [51]    2197      NA      NA      NA      NA      NA      NA      NA      NA      NA
 [61]      NA 2460375  117649   32768      NA  262144   64000  456533  912673  912673
 [71]  614125      NA    1000   19683      NA     343  110592   42875  226981  493039
 [81]  250047    4096      NA      NA  512000 1259712    8000  140608  551368  125000
 [91]  262144  205379   59319     729    4096  474552   42875  287496 1815848  704969
[101] 1331000      NA      NA   85184   21952  274625      NA   10648  205379   12167
[111]   29791   85184    9261     729      NA   91125 4741632  389017      NA  438976
[121] 1643032  592704  614125  884736  474552  389017  753571  103823   32768    8000
[131]   12167    9261   13824   85184    9261   21952     729    2197   97336    5832
[141]    2197   13824    4096    2197   12167   46656     343    2744   27000      NA
```

3.2 Other control structures (file Rcourse.4)

A useful function to establish similar control structures is ifelse(), which operates with logical conditions:

```
> ifelse(airquality$Ozone>=30,1,0)
  [1]  1  1  0  0 NA  0  0  0  0 NA  0  0  0  0  0  0  1  0  1  0  0  0  0  1 NA NA NA  0
1
 [30]  1  1 NA NA NA NA NA NA  0 NA  1  1 NA NA  0 NA NA  0  1  0  0  0 NA NA NA NA NA NA
NA
 [59] NA NA NA  1  1  1 NA  1  1  1  1  1  1 NA  0  0 NA  0  1  1  1  1  1  0 NA NA  1  1
0
 [88]  1  1  1  1  1  1  0  0  1  1  1  1  1  1 NA NA  1  0  1 NA  0  1  0  1  1  0  0 NA
1
[117]  1  1 NA  1  1  1  1  1  1  1  1  1  1  0  0  0  0  1  0  0  0  0  1  0  0  0  0  0
0
[146]  1  0  0  1 NA  0  0  0
```

3.3 "apply" family of functions (file Rcourse.4)

I use this name to refer to a very important group of functions in R. These functions allow us to perform a loop, applying a function, over one or two dimensions of an object.

They have different names depending on how they return their results, the class of the objects over which they can operate or the dimensions over which they can perform the loop (if they are uni- or multi-variate). These functions constitute one of the most convenient features of R as a programming language respect to other languages, that need more complicated loops in order to perform this kind of operations. Let's see some examples:

In a one dimensional vector, or a list, sapply() returns a vector:

```
> sapply(airquality$Wind,sqrt)
  [1] 2.720294 2.828427 3.549648 3.391165 3.781534 3.860052 2.932576 3.714835 4.483302
 [10] 2.932576 2.626785 3.114482 3.033150 3.301515 3.633180 3.391165 3.464102 4.289522
 [19] 3.391165 3.114482 3.114482 4.074310 3.114482 3.464102 4.074310 3.860052 2.828427
 [28] 3.464102 3.860052 2.387467 2.720294 2.932576 3.114482 4.012481 3.033150 2.932576
 [37] 3.781534 3.114482 2.626785 3.714835 3.391165 3.301515 3.033150 2.828427 3.714835
 [46] 3.391165 3.860052 4.549725 3.033150 3.391165 3.209361 2.509980 1.303840 2.144761
 [55] 2.509980 2.828427 2.828427 3.209361 3.391165 3.860052 2.828427 2.024846 3.033150
 [64] 3.033150 3.301515 2.144761 3.301515 2.258318 2.509980 2.387467 2.720294 2.932576
 [73] 3.781534 3.860052 3.860052 3.781534 2.626785 3.209361 2.509980 2.258318 3.391165
 [82] 2.626785 3.114482 3.391165 2.932576 2.828427 2.932576 3.464102 2.720294 2.720294
 [91] 2.720294 3.033150 2.626785 3.714835 2.720294 2.626785 2.720294 2.144761 2.000000
[100] 3.209361 2.828427 2.932576 3.391165 3.391165 3.391165 3.114482 3.391165 3.209361
[109] 2.509980 2.720294 3.301515 3.209361 3.937004 3.781534 3.549648 3.114482 1.843909
[118] 2.828427 2.387467 3.114482 1.516575 2.509980 2.509980 2.626785 2.258318 1.673320
[127] 2.144761 2.720294 3.937004 3.301515 3.209361 3.301515 3.114482 3.860052 3.937004
[136] 2.509980 3.301515 3.391165 2.626785 3.714835 3.209361 3.209361 2.828427 3.549648
[145] 3.033150 3.209361 3.209361 4.074310 2.626785 3.633180 3.781534 2.828427 3.391165
```

The equivalent to sapply() in matrices, data frames or arrays is apply():

```
> apply(airquality[,1:4],2,mean,na.rm=T)
     Ozone    Solar.R       Wind       Temp
 42.129310 185.931507   9.957516  77.882353
```

In lists, or one dimensional vectors, lapply() returns a list:

```
> newlist<-list(a=rnorm(10),b=rpois(10,3),c=rchisq(10,1))
> lapply(newlist,sd)
$a
[1] 1.375927

$b
[1] 0.8232726

$c
[1] 1.78412
```

mapply() is a multivariate version of sapply():

```
> n<-seq(10,60,10)
> m<-seq(0,25,5)
> d<-1:6
> mapply(rnorm,n=n,mean=m,sd=d)
```

```
[[1]]
 [1] -0.3410049  0.4323415  0.3516355 -0.8372031 -0.2830185  0.8749000  0.3707977
-0.5184339
 [9]  0.8477962  0.8887328

[[2]]
 [1] 6.3192104 7.0658536 5.4359676 5.8912294 2.1125294 4.3101721 6.6034879 4.7204819
 [9] 7.1181019 2.9935151 5.0502183 5.5654906 4.7421671 5.7359081 2.8021586 4.4425867
[17] 6.3686981 0.5284532 4.3926148 1.2366614

[[3]]
 [1]  9.262279  8.622019  4.181492  7.615304 14.884560  4.560162  5.584390 11.926778
 [9]  9.866775 11.638280 11.613373 11.247880  5.734978  7.427800 10.323455  8.686091
[17]  7.926096 12.337703  6.921334  8.553245  7.993458 11.030824 10.491214  6.516386
[25] 10.415909 14.582014 12.652237 10.320437 14.636407  9.161725

[[4]]
 [1] 14.039670 17.025962 16.330808 12.575764 16.764375 16.456429 15.978264 12.895216
 [9] 12.527887 13.365563 17.966810  5.874060 22.637297 22.339022 16.317141 22.704709
[17] 17.749680 16.412733 14.430827 11.010198 18.478512 11.825207 10.974432  6.948024
[25] 15.155345 10.159393 11.055073 12.300910 13.175993 10.230596 17.930956 11.169400
[33] 10.969809 15.216220 20.892360 17.753215 14.092371 15.634261 19.314253 21.098948

[[5]]
 [1] 18.682156 25.094968  9.274294 12.397940 16.868750 21.762478 20.529611 24.467231
 [9] 24.814708 27.493695 16.112136 19.645944 20.931607 13.369234 14.130686 17.012648
[17] 26.948434 25.387613 22.106181 10.389036 23.513035 19.190013 26.861857 26.922371
[25] 16.230487 17.304604 17.440033 16.025489 16.709490 19.859731 23.525476 16.700985
[33] 16.405295 23.698648 23.201127 17.003933 23.043499 16.229006  9.481021 31.575795
[41] 29.347441 22.074244 27.272004 13.222210 22.353553 18.358591 15.424012 26.625223
[49] 26.432720 19.813154

[[6]]
 [1] 26.49586 17.29606 16.81156 28.34440 27.03726 23.52428 30.28758 29.74780 27.61815
[10] 10.53872 28.55658 29.47653  8.09089 21.52203 30.08308 18.76776 22.71881 22.68679
[19] 26.34976 22.48113 22.39868 24.56411 11.43283 24.82806 29.38269 19.69017 25.42932
[28] 35.88416 32.38297 28.33910 25.45460 22.11562 29.26370 29.24052 33.44069 20.57193
[37] 28.97481 26.70283 34.14409 25.79775 12.72875 18.82033 22.87307 14.07122 21.73772
[46] 16.40428 31.44385 27.00159 24.90420 16.25908 28.38914 24.12845 14.76581 20.14278
[55] 17.80610 25.30808 15.69419 35.68506 15.53320 30.82491
```

Another one, tapply(), very similar to by() function, is very useful for applying functions to data frames structured in groups, categories or factors, like *iris*:

```
> data(iris)
> tapply(iris$Sepal.Width,iris$Species,sd)
    setosa versicolor  virginica
 0.3790644  0.3137983  0.3224966
```

Other functions of this “family” that we are just going to mention are: vapply(), eapply(), rapply().

There are other functions that perform similar operations. Let's see one important example: the function aggregate(). This function can perform a similar task than tapply() or by(), but with several factors and variables at once, which is s great advantage. Here is an example applying a function that returns most of the basic descriptive statistics at once:

```
> require(pastecs)
Loading required package: pastecs
Loading required package: boot
> data(CO2)
> aggregate(CO2[,4:5],by=list(CO2$Plant,CO2$Type),FUN=stat.desc)
   Group.1     Group.2 conc.nbr.val conc.nbr.null  conc.nbr.na     conc.min
1      Qn1      Quebec 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
2      Qn2      Quebec 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
3      Qn3      Quebec 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
4      Qc1      Quebec 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
5      Qc3      Quebec 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
6      Qc2      Quebec 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
7      Mn3 Mississippi 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
8      Mn2 Mississippi 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
9      Mn1 Mississippi 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
10     Mc2 Mississippi 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
11     Mc3 Mississippi 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
12     Mc1 Mississippi 7.000000e+00  0.000000e+00 0.000000e+00 9.500000e+01
       conc.max   conc.range     conc.sum  conc.median    conc.mean
1  1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
2  1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
3  1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
4  1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
5  1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
6  1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
7  1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
8  1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
9  1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
10 1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
11 1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
12 1.000000e+03 9.050000e+02 3.045000e+03 3.500000e+02 4.350000e+02
   conc.SE.mean conc.CI.mean.0.95     conc.var conc.std.dev conc.coef.var
1  1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
2  1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
3  1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
4  1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
5  1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
6  1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
7  1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
8  1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
9  1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
10 1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
11 1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
12 1.200893e+02      2.938478e+02 1.009500e+05 3.177263e+02  7.304053e-01
   uptake.nbr.val uptake.nbr.null uptake.nbr.na  uptake.min  uptake.max
1       7.0000000       0.0000000     0.0000000  16.0000000  39.7000000
2       7.0000000       0.0000000     0.0000000  13.6000000  44.3000000
3       7.0000000       0.0000000     0.0000000  16.2000000  45.5000000
4       7.0000000       0.0000000     0.0000000  14.2000000  38.7000000
5       7.0000000       0.0000000     0.0000000  15.1000000  41.4000000
6       7.0000000       0.0000000     0.0000000   9.3000000  42.4000000
7       7.0000000       0.0000000     0.0000000  11.3000000  28.5000000
8       7.0000000       0.0000000     0.0000000  12.0000000  32.4000000
```

```
9       7.0000000       0.0000000     0.0000000  10.6000000  35.5000000
10      7.0000000       0.0000000     0.0000000   7.7000000  14.4000000
11      7.0000000       0.0000000     0.0000000  10.6000000  19.9000000
12      7.0000000       0.0000000     0.0000000  10.5000000  22.2000000
   uptake.range  uptake.sum uptake.median uptake.mean uptake.SE.mean
1    23.7000000 232.6000000    35.3000000  33.2285714      3.1048897
2    30.7000000 246.1000000    40.6000000  35.1571429      4.1591470
3    29.3000000 263.3000000    42.1000000  37.6142857      3.9119127
4    24.5000000 209.8000000    32.5000000  29.9714286      3.1501863
5    26.3000000 228.1000000    38.1000000  32.5857143      3.9010028
6    33.1000000 228.9000000    37.5000000  32.7000000      4.2849682
7    17.2000000 168.8000000    27.8000000  24.1142857      2.4509890
8    20.4000000 191.4000000    31.1000000  27.3428571      2.8925073
9    24.9000000 184.8000000    30.0000000  26.4000000      3.2861180
10    6.7000000  85.0000000    12.5000000  12.1428571      0.8265986
11    9.3000000 121.1000000    17.9000000  17.3000000      1.1524301
12   11.7000000 126.0000000    18.9000000  18.0000000      1.5567059
   uptake.CI.mean.0.95  uptake.var uptake.std.dev uptake.coef.var
1            7.5973914  67.4823810      8.2147660       0.2472200
2           10.1770660 121.0895238     11.0040685       0.3129967
3            9.5721056 107.1214286     10.3499482       0.2751600
4            7.7082281  69.4657143      8.3346094       0.2780852
5            9.5454101 106.5247619     10.3210834       0.3167364
6           10.4849394 128.5266667     11.3369602       0.3466960
7            5.9973540  42.0514286      6.4847073       0.2689156
8            7.0777105  58.5661905      7.6528551       0.2798850
9            8.0408410  75.5900000      8.6942510       0.3293277
10           2.0226140   4.7828571      2.1869744       0.1801038
11           2.8198950   9.2966667      3.0490436       0.1762453
12           3.8091222  16.9633333      4.1186567       0.2288143
```

Section 4: Managing data

Control

4.1 Importing data into R (file Rcourse.5)

One of the first important things that you may have to do in an R session is importing data files. In case of having few data, you may directly type them and store them as a vector or matrix objects. In some of the GUIs, there is a data editors available from a menu. But the most common way to load data in your session is to import them from diverse file formats. The most common of those are the txt, csv, xls, xlsx, … R also admit imports from Arff, DBF, SPSS, SAS, Minitab, Stata... A few, but important functions to import these file formats (txt, csv) are present in the default packages. For the other formats, you will need to install specific packages (*gdata*, *xlsx*, *RODBC*, *Hmisc*, *foreign*). The general structure of the functions and main argument is: read.fileformat("filename.extension"). So let's try a few examples:
Import from txt the "Growth" data file provided to you in the course folder:

```
> dat1<-read.table("Growth.txt")
> head(dat1)
```

Import from csv:

```
> dat2<-read.csv2("Growth.csv",sep=",")
> head(dat2)
```

Here we introduce an important argument of these functions: "sep".
In the csv, the symbol separating fields may vary between USA, Europe, as well as different operating systems. The csv format is also different between USA and Europe, and hence, the import function. In the USA, it is better to use read.csv() and in Europe read.csv2().
csv is my preferred format for data import, but it is a bit tricky. You have to transform a xls or xlsx file into csv through the Office command "save as", where you can select among several formats, among them, the csv. The problem is that this csv file will store in its memory those cells were you have deleted data or text as if they had missing data. Then, in your R session, those cells will be assigned an NA. These NAs might screw your further analysis at some point. The solution is when you have a xls or xlsx file with your data correctly organized, copy them and paste them in a new file of the same format. Then, without altering their cell contents, save it as csv.

Import from xls:

```
> require(gdata)
> dat3<-read.xls("Growth.xls")
> head(dat3)
```

Import from xlsx:

```
> require(xlsx)
> dat4<-read.xlsx("Growth.xlsx",1)
> head(dat4)
```

In this case, it is necessary to indicate, with the argument "sheetIndex", the page number to import from the file.

Actually, with the function read.xls() from package *gdata*, you can also import xlsx format (this was not possible with precursors of read.xls()):

```
> dat5<-read.xls("Growth.xlsx")
> head(dat5)
```

If the files to import are not located in your working directory, you will need to write the whole directory path before the file name:

```
> dat<-("/path/name of the file.extension") #Linux and MacOS
> dat<-("C:/path/name of the file.extension") #Windows
```

It is also possible to import a file from the internet:

```
> dat<-read.table(file="http://www.mypage.com/file.txt")
```

4.2 Exporting data (file Rcourse.5)

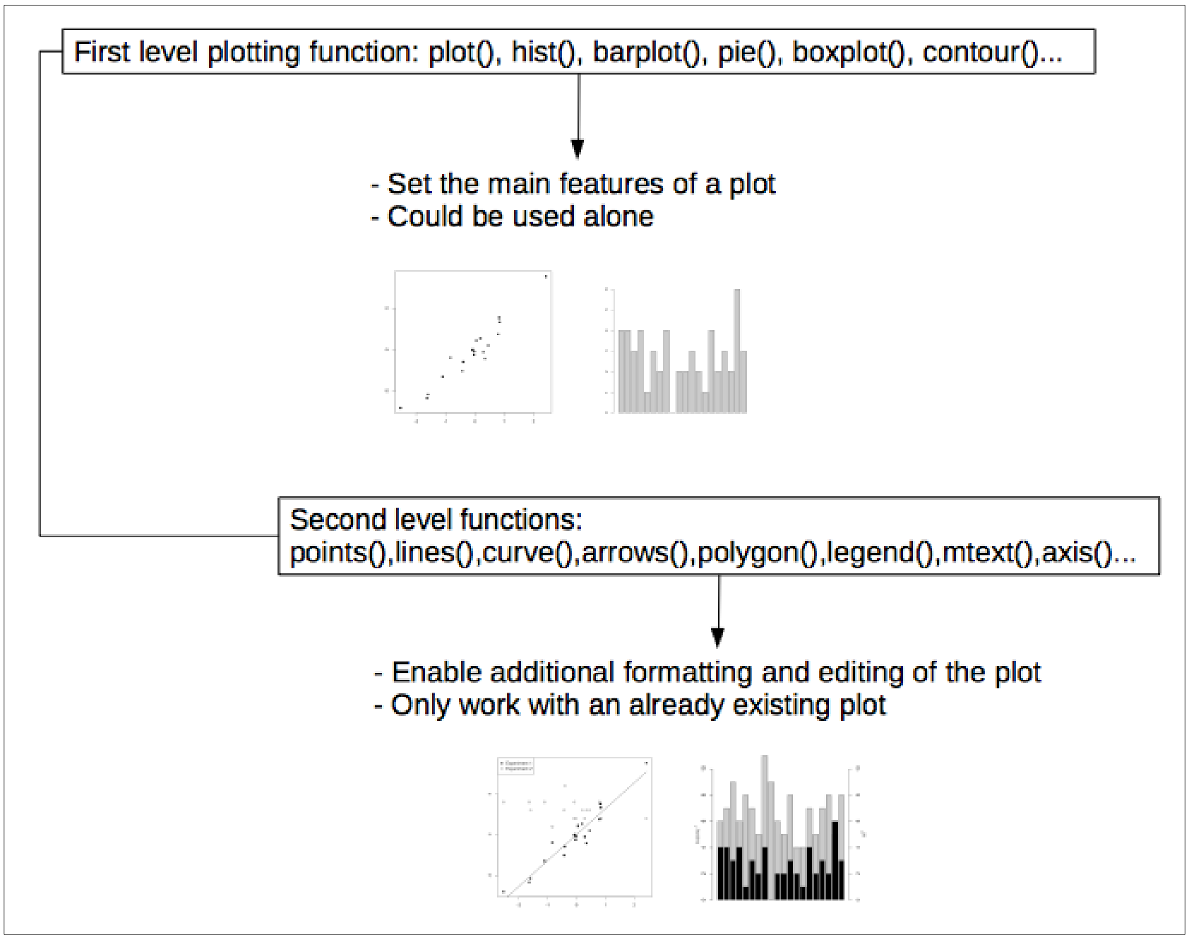
The functions for exporting data are analogous to those for importing data, both in the name and the file formats available. The exported files will go to the working directory. The main function arguments are the name of the object to export and the exported file name with the extension:

```
> write.table(iris,"iris.txt")
> write.csv2(iris,"iris.csv")
> require(gdata)
> write.fwf(iris,"iris.xls")
> require(xlsx)
> write.xlsx(iris,"iris.xlsx")
```

Section 5: Basic features of functions from *graphics* package. Other graphical packages

Basic examples of graphical functions **(file Rcourse.6)**

In the package *graphics*, installed and loaded by default, there is already a great variety of graphical functions. A nice overview can be seen with the demo of this package, as we mentioned already in the section above. The most basic function is plot(). With this one you can get already many different plots. But there are also other relevant functions: hist(), barplot(), boxplot() (the one from our example above), pie(), persp(), contour(). These are all first level graphical functions. They create the general features of a specific kind of plot: dispersion, histogram, bars, pie, three dimensional perspective, etc. additionally, there are other functions of second level that are also very important, because they are used to add secondary features to an already existing plot: mtext() (adds text in the margins), legend() (adds a legend), points() (adds additional data), curve() (adds a fitted curve), lines() (adds fitted lines), arrows() (adds arrows), polygon() (adds a polygon), etc. This is a diagram of this way of structuring a plot code:



First level plotting function: plot(), hist(), barplot(), pie(), boxplot(), contour()...
- Set the main features of a plot
- Could be used alone
Second level functions:
points(),lines(),curve(),arrows(),polygon(),legend(),mtext(),axis()...
- Enable additional formatting and editing of the plot
- Only work with an already existing plot

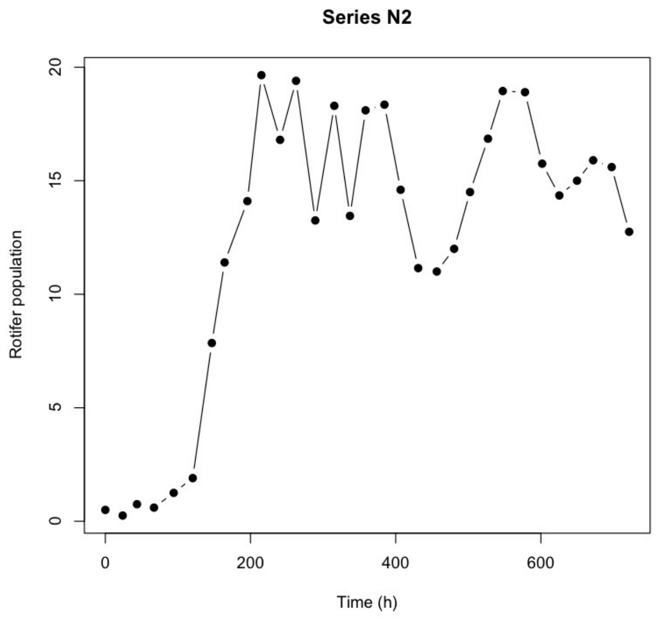
Remember that you can always check all the functions in a package in detail at the pdf manual from the R project web site. And also by typing:

```
> help(package="graphics")
```

Now we are going to see just a few examples of the use of remarkable functions from this

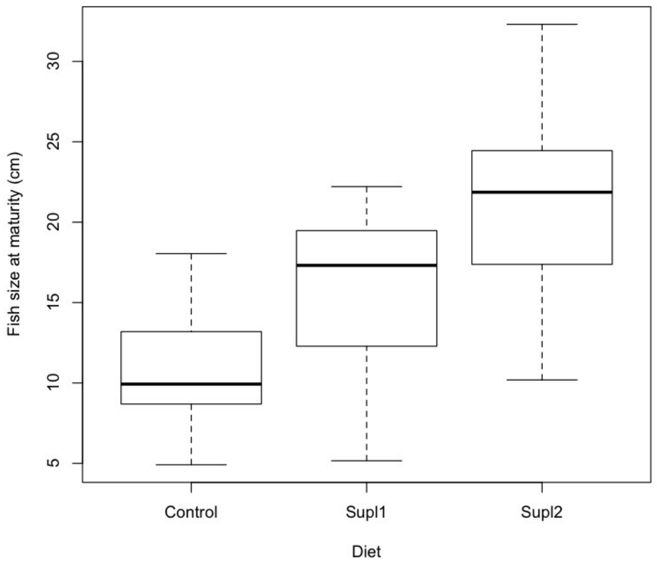
group.
The plot() function already enables creating many different kinds of plots, like scatterplots, lines, lines plus dots. This is a basic example of how to use this function:

```
> #Plot
> data<-read.csv2("SeriesN2.csv",sep=",")
> plot(data[,1],data[,2],type="b",pch=19,xlab="Time (h)",ylab="Rotifer
population",main="Series N2")
```



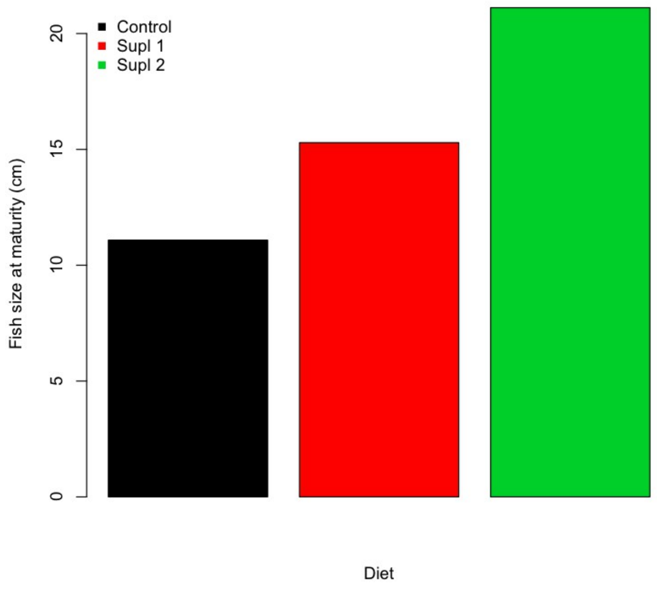
Here, an example with the box – whiskers function boxplot():

```
> #Boxplot
> data2<-read.csv2("FishSize.csv")
> boxplot(data2$Size~data2$Treatment,xlab="Diet",ylab="Fish size at maturity
(cm)")
```



And this, with the barplot() function, introducing now a second – level funtion, the function legend():

```
> #Barplot
> Size<-tapply(data2$Size,data2$Treatment,mean)
> barplot(Size,col=c(1,2,3),xlab="Diet",ylab="Fish size at maturity
(cm)",axisnames=F)
> legend("topleft",c("Control","Supl 1","Supl
2"),pch=c(15,15,15),col=c(1,2,3),bty="n")
```



In order to select the colors for any graphical device, the argument “col” has a numeric code, each number representing a specific color. But the colors could be specified by a character stringl which is its own name. But for small differences in color it is difficult to remember. You can check this with the function colors:

```
> colors()
  [1] "white"                "aliceblue"            "antiquewhite"
  [4] "antiquewhite1"        "antiquewhite2"        "antiquewhite3"
  [7] "antiquewhite4"        "aquamarine"           "aquamarine1"
 [10] "aquamarine2"          "aquamarine3"          "aquamarine4"
 [13] "azure"                "azure1"               "azure2"
 [16] "azure3"               "azure4"               "beige"
 [19] "bisque"               "bisque1"              "bisque2"
 [22] "bisque3"              "bisque4"              "black"
 [25] "blanchedalmond"       "blue"                 "blue1"
 [28] "blue2"                "blue3"                "blue4"
 [31] "blueviolet"           "brown"                "brown1"
 [34] "brown2"               "brown3"               "brown4"
 [37] "burlywood"            "burlywood1"           "burlywood2"
 [40] "burlywood3"           "burlywood4"           "cadetblue"
 [43] "cadetblue1"           "cadetblue2"           "cadetblue3"
 [46] "cadetblue4"          "chartreuse"          "chartreuse1"
 [49] "chartreuse2"         "chartreuse3"         "chartreuse4"
 [52] "chocolate"           "chocolate1"          "chocolate2"
 [55] "chocolate3"          "chocolate4"          "coral"
 [58] "coral1"              "coral2"              "coral3"
 [61] "coral4"              "cornflowerblue"      "cornsilk"
 [64] "cornsilk1"           "cornsilk2"           "cornsilk3"
 [67] "cornsilk4"           "cyan"                "cyan1"
 [70] "cyan2"               "cyan3"               "cyan4"
 [73] "darkblue"            "darkcyan"            "darkgoldenrod"
 [76] "darkgoldenrod1"      "darkgoldenrod2"      "darkgoldenrod3"
 [79] "darkgoldenrod4"      "darkgray"            "darkgreen"
 [82] "darkgrey"            "darkkhaki"           "darkmagenta"
 [85] "darkolivegreen"      "darkolivegreen1"     "darkolivegreen2"
 [88] "darkolivegreen3"     "darkolivegreen4"     "darkorange"
 [91] "darkorange1"         "darkorange2"         "darkorange3"
 [94] "darkorange4"         "darkorchid"          "darkorchid1"
 [97] "darkorchid2"         "darkorchid3"         "darkorchid4"
[100] "darkred"             "darksalmon"          "darkseagreen"
[103] "darkseagreen1"       "darkseagreen2"       "darkseagreen3"
[106] "darkseagreen4"       "darkslateblue"       "darkslategray"
[109] "darkslategray1"      "darkslategray2"      "darkslategray3"
[112] "darkslategray4"      "darkslategrey"       "darkturquoise"
[115] "darkviolet"          "deeppink"            "deeppink1"
[118] "deeppink2"           "deeppink3"           "deeppink4"
[121] "deepskyblue"         "deepskyblue1"        "deepskyblue2"
[124] "deepskyblue3"        "deepskyblue4"        "dimgray"
[127] "dimgrey"             "dodgerblue"          "dodgerblue1"
[130] "dodgerblue2"         "dodgerblue3"         "dodgerblue4"
[133] "firebrick"           "firebrick1"          "firebrick2"
[136] "firebrick3"          "firebrick4"          "floralwhite"
[139] "forestgreen"         "gainsboro"           "ghostwhite"
[142] "gold"                "gold1"               "gold2"
[145] "gold3"               "gold4"               "goldenrod"
[148] "goldenrod1"          "goldenrod2"          "goldenrod3"
[151] "goldenrod4"          "gray"                "gray0"
[154] "gray1"               "gray2"               "gray3"
[157] "gray4"               "gray5"               "gray6"
[160] "gray7"               "gray8"               "gray9"
[163] "gray10"              "gray11"              "gray12"
[166] "gray13"              "gray14"              "gray15"
[169] "gray16"              "gray17"              "gray18"
[172] "gray19"              "gray20"              "gray21"
[175] "gray22"              "gray23"              "gray24"
[178] "gray25"              "gray26"              "gray27"
[181] "gray28"              "gray29"              "gray30"
[184] "gray31"              "gray32"              "gray33"
[187] "gray34"              "gray35"              "gray36"
[190] "gray37"              "gray38"              "gray39"
[193] "gray40"              "gray41"              "gray42"
[196] "gray43"              "gray44"              "gray45"
[199] "gray46"              "gray47"              "gray48"
[202] "gray49"              "gray50"              "gray51"
[205] "gray52"              "gray53"              "gray54"
[208] "gray55"              "gray56"              "gray57"
[211] "gray58"              "gray59"              "gray60"
```

```
[214] "gray61"               "gray62"               "gray63"
[217] "gray64"               "gray65"               "gray66"
[220] "gray67"               "gray68"               "gray69"
[223] "gray70"               "gray71"               "gray72"
[226] "gray73"               "gray74"               "gray75"
[229] "gray76"               "gray77"               "gray78"
[232] "gray79"               "gray80"               "gray81"
[235] "gray82"               "gray83"               "gray84"
[238] "gray85"               "gray86"               "gray87"
[241] "gray88"               "gray89"               "gray90"
[244] "gray91"               "gray92"               "gray93"
[247] "gray94"               "gray95"               "gray96"
[250] "gray97"               "gray98"               "gray99"
[253] "gray100"              "green"                "green1"
[256] "green2"               "green3"               "green4"
[259] "greenyellow"          "grey"                 "grey0"
[262] "grey1"                "grey2"                "grey3"
[265] "grey4"                "grey5"                "grey6"
[268] "grey7"                "grey8"                "grey9"
[271] "grey10"               "grey11"               "grey12"
[274] "grey13"               "grey14"               "grey15"
[277] "grey16"               "grey17"               "grey18"
[280] "grey19"               "grey20"               "grey21"
[283] "grey22"               "grey23"               "grey24"
[286] "grey25"               "grey26"               "grey27"
[289] "grey28"               "grey29"               "grey30"
[292] "grey31"               "grey32"               "grey33"
[295] "grey34"               "grey35"               "grey36"
[298] "grey37"               "grey38"               "grey39"
[301] "grey40"               "grey41"               "grey42"
[304] "grey43"               "grey44"               "grey45"
[307] "grey46"               "grey47"               "grey48"
[310] "grey49"               "grey50"               "grey51"
[313] "grey52"               "grey53"               "grey54"
[316] "grey55"               "grey56"               "grey57"
[319] "grey58"               "grey59"               "grey60"
[322] "grey61"               "grey62"               "grey63"
[325] "grey64"               "grey65"               "grey66"
[328] "grey67"               "grey68"               "grey69"
[331] "grey70"               "grey71"               "grey72"
[334] "grey73"               "grey74"               "grey75"
[337] "grey76"               "grey77"               "grey78"
[340] "grey79"               "grey80"               "grey81"
[343] "grey82"               "grey83"               "grey84"
[346] "grey85"               "grey86"               "grey87"
[349] "grey88"               "grey89"               "grey90"
[352] "grey91"               "grey92"               "grey93"
[355] "grey94"               "grey95"               "grey96"
[358] "grey97"               "grey98"               "grey99"
[361] "grey100"              "honeydew"             "honeydew1"
[364] "honeydew2"            "honeydew3"            "honeydew4"
[367] "hotpink"              "hotpink1"             "hotpink2"
[370] "hotpink3"             "hotpink4"             "indianred"
[373] "indianred1"           "indianred2"           "indianred3"
[376] "indianred4"           "ivory"                "ivory1"
[379] "ivory2"               "ivory3"               "ivory4"
```

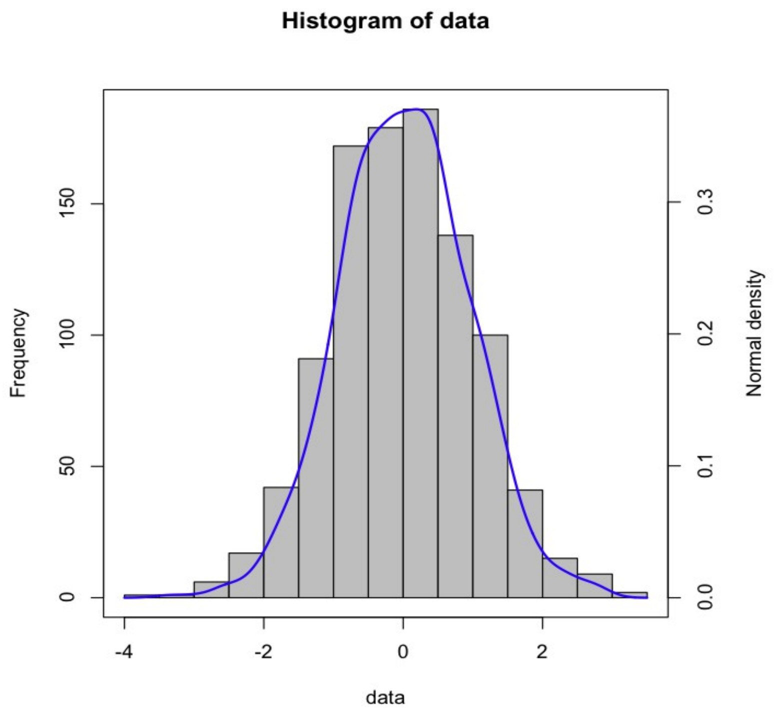
```
[382] "khaki"                "khaki1"               "khaki2"
[385] "khaki3"               "khaki4"               "lavender"
[388] "lavenderblush"        "lavenderblush1"       "lavenderblush2"
[391] "lavenderblush3"       "lavenderblush4"       "lawngreen"
[394] "lemonchiffon"         "lemonchiffon1"        "lemonchiffon2"
[397] "lemonchiffon3"        "lemonchiffon4"        "lightblue"
[400] "lightblue1"           "lightblue2"           "lightblue3"
[403] "lightblue4"           "lightcoral"           "lightcyan"
[406] "lightcyan1"           "lightcyan2"           "lightcyan3"
[409] "lightcyan4"           "lightgoldenrod"       "lightgoldenrod1"
[412] "lightgoldenrod2"      "lightgoldenrod3"      "lightgoldenrod4"
[415] "lightgoldenrodyellow" "lightgray"            "lightgreen"
[418] "lightgrey"            "lightpink"            "lightpink1"
[421] "lightpink2"           "lightpink3"           "lightpink4"
[424] "lightsalmon"          "lightsalmon1"         "lightsalmon2"
[427] "lightsalmon3"         "lightsalmon4"         "lightseagreen"
[430] "lightskyblue"         "lightskyblue1"        "lightskyblue2"
[433] "lightskyblue3"        "lightskyblue4"        "lightslateblue"
[436] "lightslategray"       "lightslategrey"       "lightsteelblue"
[439] "lightsteelblue1"      "lightsteelblue2"      "lightsteelblue3"
[442] "lightsteelblue4"      "lightyellow"          "lightyellow1"
[445] "lightyellow2"         "lightyellow3"         "lightyellow4"
[448] "limegreen"            "linen"                "magenta"
[451] "magenta1"             "magenta2"             "magenta3"
[454] "magenta4"             "maroon"               "maroon1"
[457] "maroon2"              "maroon3"              "maroon4"
[460] "mediumaquamarine"     "mediumblue"           "mediumorchid"
[463] "mediumorchid1"        "mediumorchid2"        "mediumorchid3"
[466] "mediumorchid4"        "mediumpurple"         "mediumpurple1"
[469] "mediumpurple2"        "mediumpurple3"        "mediumpurple4"
[472] "mediumseagreen"       "mediumslateblue"      "mediumspringgreen"
[475] "mediumturquoise"      "mediumvioletred"      "midnightblue"
[478] "mintcream"            "mistyrose"            "mistyrose1"
[481] "mistyrose2"           "mistyrose3"           "mistyrose4"
[484] "moccasin"             "navajowhite"          "navajowhite1"
[487] "navajowhite2"         "navajowhite3"         "navajowhite4"
[490] "navy"                 "navyblue"             "oldlace"
[493] "olivedrab"            "olivedrab1"           "olivedrab2"
[496] "olivedrab3"           "olivedrab4"           "orange"
[499] "orange1"              "orange2"              "orange3"
[502] "orange4"              "orangered"            "orangered1"
[505] "orangered2"           "orangered3"           "orangered4"
[508] "orchid"               "orchid1"              "orchid2"
[511] "orchid3"              "orchid4"              "palegoldenrod"
[514] "palegreen"            "palegreen1"           "palegreen2"
[517] "palegreen3"           "palegreen4"           "paleturquoise"
[520] "paleturquoise1"       "paleturquoise2"       "paleturquoise3"
[523] "paleturquoise4"       "palevioletred"        "palevioletred1"
[526] "palevioletred2"       "palevioletred3"       "palevioletred4"
[529] "papayawhip"           "peachpuff"            "peachpuff1"
[532] "peachpuff2"           "peachpuff3"           "peachpuff4"
[535] "peru"                 "pink"                 "pink1"
[538] "pink2"                "pink3"                "pink4"
[541] "plum"                 "plum1"                "plum2"
[544] "plum3"                "plum4"                "powderblue"
[547] "purple"               "purple1"              "purple2"
```

```
[550] "purple3"              "purple4"              "red"
[553] "red1"                 "red2"                 "red3"
[556] "red4"                 "rosybrown"            "rosybrown1"
[559] "rosybrown2"           "rosybrown3"           "rosybrown4"
[562] "royalblue"            "royalblue1"           "royalblue2"
[565] "royalblue3"           "royalblue4"           "saddlebrown"
[568] "salmon"               "salmon1"              "salmon2"
[571] "salmon3"              "salmon4"              "sandybrown"
[574] "seagreen"             "seagreen1"            "seagreen2"
[577] "seagreen3"            "seagreen4"            "seashell"
[580] "seashell1"            "seashell2"            "seashell3"
[583] "seashell4"            "sienna"               "sienna1"
[586] "sienna2"              "sienna3"              "sienna4"
[589] "skyblue"              "skyblue1"             "skyblue2"
[592] "skyblue3"             "skyblue4"             "slateblue"
[595] "slateblue1"           "slateblue2"           "slateblue3"
[598] "slateblue4"           "slategray"            "slategray1"
[601] "slategray2"           "slategray3"           "slategray4"
[604] "slategrey"            "snow"                 "snow1"
[607] "snow2"                "snow3"                "snow4"
[610] "springgreen"          "springgreen1"         "springgreen2"
[613] "springgreen3"         "springgreen4"         "steelblue"
[616] "steelblue1"           "steelblue2"           "steelblue3"
[619] "steelblue4"           "tan"                  "tan1"
[622] "tan2"                 "tan3"                 "tan4"
[625] "thistle"              "thistle1"             "thistle2"
[628] "thistle3"             "thistle4"             "tomato"
[631] "tomato1"              "tomato2"              "tomato3"
[634] "tomato4"              "turquoise"            "turquoise1"
[637] "turquoise2"           "turquoise3"           "turquoise4"
[640] "violet"               "violetred"            "violetred1"
[643] "violetred2"           "violetred3"           "violetred4"
[646] "wheat"                "wheat1"               "wheat2"
[649] "wheat3"               "wheat4"               "whitesmoke"
[652] "yellow"               "yellow1"              "yellow2"
[655] "yellow3"              "yellow4"              "yellowgreen"
```

Another typical plot are the histograms. We can plot, for instance, a simulated normal distribution with a non – parametric density curve. We first create 1000 random data drawn from a normal distribution with 0 mean and 1 standard deviation, then we fit a gaussian kernel estimator of its density. We enlarge the plot margins with the argument mar inside the function par() so that we can suitably fit the axis and text. Then we plot the density and add a second axis for the density scale:

```
> data<-rnorm(1000)
> fit<-density(data)
> par(mar=c(6,6,6,6))
> hist(data,col="grey")
> par(new=TRUE)
> plot(fit$y,type="l",lwd=2,col="blue",xlab="",ylab="",xaxt="n",yaxt="n")
> axis(4)
> mtext("Normal density",4,3,at=0.2)
```

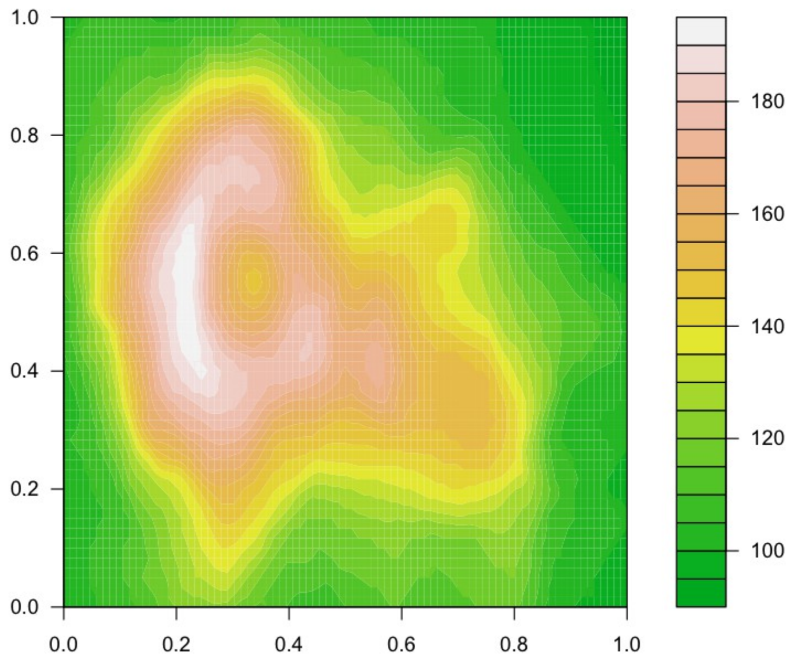
And here it is how the plot looks like:



There are also available functions to generate plots of surfaces, like contour() or filled.contour(). The data files could be organize in different ways. In this example, rows are oriented east to west, and columns south to north, so cells are contiguous topological points. The numerical values of each cell correspond with a level in the scale.

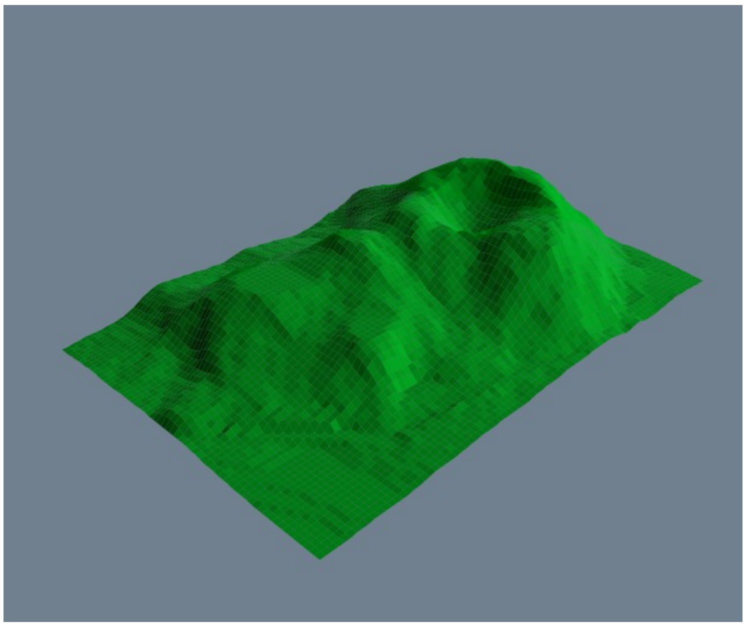
```
> head(volcano)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]  100  100  101  101  101  101  101  100  100   100   101   101
[2,]  101  101  102  102  102  102  102  101  101   101   102   102
[3,]  102  102  103  103  103  103  103  102  102   102   103   103
[4,]  103  103  104  104  104  104  104  103  103   103   103   104
[5,]  104  104  105  105  105  105  105  104  104   103   104   104
[6,]  105  105  105  106  106  106  106  105  105   104   104   105
     [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23]
[1,]   102   102   102   102   103   104   103   102   101   101   102
[2,]   103   103   103   103   104   105   104   103   102   102   103
[3,]   104   104   104   104   105   106   105   104   104   105   106
[4,]   104   104   105   105   106   107   106   106   106   107   108
[5,]   105   105   105   106   107   108   108   108   109   110   112
[6,]   105   106   106   107   109   110   110   112   113   115   116
     [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34]
[1,]   103   104   104   105   107   107   107   108   108   110   110
[2,]   105   106   106   107   109   110   110   110   110   111   112
[3,]   107   108   110   111   113   114   115   114   115   116   118
[4,]   110   111   114   117   118   117   119   120   121   122   124
[5,]   114   115   118   121   122   121   123   128   131   129   130
[6,]   118   119   121   124   126   126   129   134   137   137   136
     [,35] [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45]
```

```
[1,]   110   110   110   110   110   110   108   108   108   107   107
[2,]   113   114   116   115   114   112   110   110   110   109   108
[3,]   119   119   121   121   120   118   116   114   112   111   110
[4,]   125   126   127   127   126   124   122   120   117   116   113
[5,]   131   131   132   132   131   130   128   126   122   119   115
[6,]   136   135   136   136   136   135   133   129   126   122   118
      [,46] [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56]
[1,]   108   108   108   108   108   107   107   107   107   106   106
[2,]   109   109   109   109   108   108   108   108   107   107   106
[3,]   110   110   110   109   109   109   109   108   108   107   107
[4,]   111   110   110   110   109   109   109   109   108   108   107
[5,]   114   112   110   110   110   110   110   109   109   108   107
[6,]   116   115   113   111   110   110   110   110   109   108   108
      [,57] [,58] [,59] [,60] [,61]
[1,]   105   105   104   104   103
[2,]   106   105   105   104   104
[3,]   106   106   105   105   104
[4,]   107   106   106   105   105
[5,]   107   107   106   106   105
[6,]   108   107   107   106   106
> filled.contour(volcano,color.palet=terrain.colors)
```



There are as well many functions that can generate 3D plots. Among them, persp() is present in the *graphics* package:

```
> z<-2*volcano
> x<-10*(1:nrow(z))
> y<-10*(1:ncol(z))
> par(bg="slategray")
> persp(x,y,z,theta=135,phi=30,col="green3",scale=FALSE,ltheta=-
120,shade=0.75,border= NA,box=FALSE)
```



A more complicated example **(file Rcourse.7)**

Now, we will edit a complex plot of a predator – prey time series in continuous culture. The predator is a rotifer and the prey a microalgae. In the plot we also represent the dilution rate of the system, the rotifer egg production and the level of defense (aggregate size) of the algae. These details are unimportant. Here, you have to learn particularly the following details: how we can add deviation bars with a specific function. How to add several data series. How to add extra axis and margin test, how to add a legend, etc. We start right after loading the data and defining a user – defined error bar function. So the first thing we do is to use the function par(), which is a function for general format issues of the plot area. Here we set the value for the "outer margins" of the plot, with the argument *oma*, the margins of the plot, with *mar*, and the possibility of plotting outside the plot area with *xpd*:

```
> par(oma=c(2,4,2,11),mar=c(5.1,4.1,6,2.1),xpd=TRUE)
```

In a second step, we introduce the rotifer data and its standard deviation bar:

```
> plot(tm,N2[,"Brach"],pch=19,xlab="Time (days)",ylab="",ylim=c(0,20))
> error.bar(tm,N2[,"Brach"],N2[,"SdBrach"])
```

Then, we create (with the function spline()) and plot (with the function points()) a non – parametric regression, with a spline method. This will show a smoothed curve of the data:

```
> spl1<-spline(tm,N2[,"Brach"],n=20*length(N2))
> points(spl1$x,spl1$y,type="l",lwd=2)
```

In the next step, we add text in the margin of the rotifer axis, with the function mtext(). Note how we use the function expression() for writing in mathematical notation and the function paste() to paste different text elements:

```
> mtext(expression(paste("Rotifers ","(ind mL"^"-1",")")),2,2,at=10)
```

Now, we need to overlay a second plot with the same structure as the previous, but showing the microalgae data. We can not plotting functions of first level together. They would not overlay automatically, rather, the actual would delete the previous. In order to overlay them, we need again to indicate, with the function par() that we are going to create a new plot over the previous one:

```
> par(new=TRUE)
```

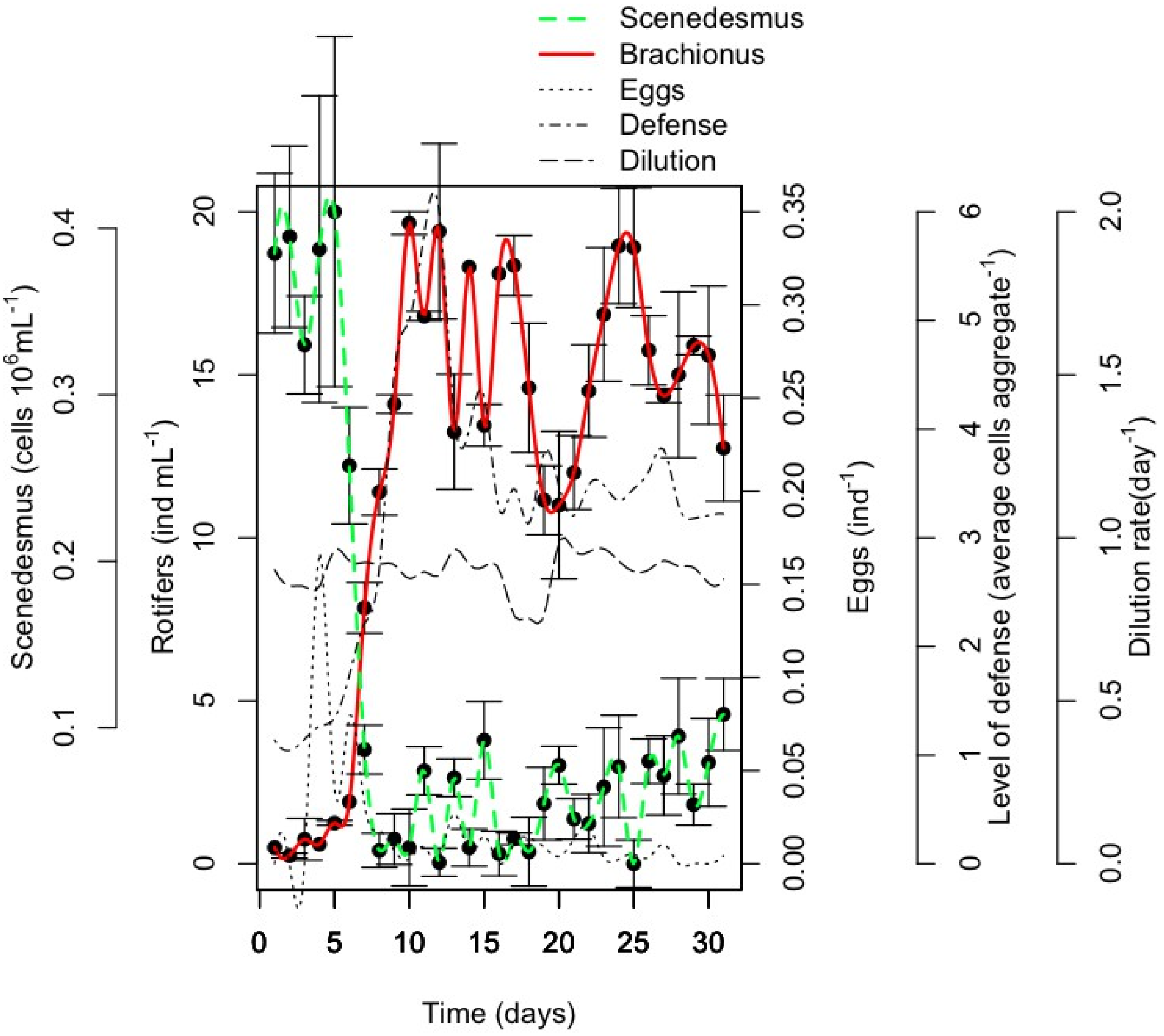
The rest of the code is simply a repetition of this same structure in order to add all the new components. But note the use of the function axis() in order to add supplementary axis:

```
> plot(tm,N2[,"Scen"],pch=19,xlab="",ylab="",yaxt="n")
> error.bar(tm,N2[,"Scen"],N2[,"SdScen"])
> spl2<-spline(tm,N2[,"Scen"],n=20*length(N2))
> points(spl2$x,spl2$y,type="l",lty=2,lwd=2,col="green")
> axis(2,line=4)
> mtext(expression(paste("Scenedesmus (cells ",10^6,"mL"^"-1",")")),2,6,at=0.25)
> par(new=TRUE)
> spl3<-spline(tm,N2[,"Eggs"],n=20*length(N2))
> plot(spl3$x,spl3$y,type="l",lty=3,lwd=1,xlab="",ylab="",yaxt="n",ylim=c(0,0.35))
> axis(4)
> mtext(expression(paste("Eggs (ind"^"-1",")")),4,3,at=0.175)
> par(new=TRUE)
> spl4<-spline(tm,N2[,"Def"],n=20*length(N2))
> plot(spl4$x,spl4$y,type="l",lty=4,lwd=1,xlab="",ylab="",yaxt="n",ylim=c(0,6))
> axis(4,line=5)
> mtext(expression(paste("Level of defense (average cells aggregate"^"-1",")")),4,7,at=3)
> par(new=TRUE)
> spl5<-spline(tm,D[,"N2"],n=20*length(D))
> plot(spl5$x,spl5$y,type="l",lty=5,lwd=1,xlab="",ylab="",xaxt="n",yaxt="n",ylim=c(0,2))
> axis(4,line=9)
> mtext(expression(paste("Dilution rate","(day"^"-1",")")),4,11,at=1)
```

And last, we add a legend to this plot, with the second level function legend():

```
> legend(x=17,y=2.7,c("Scenedesmus","Brachionus","Eggs","Defense","Dilution"),
lty=c(2,1,3,4,5),col=c(“green”,”red”,”black”,”black”,”black”),lwd=c(2,2,1,1,1),bty=”n”)
```

And this is how the final plot looks like:



* There are many possibilities for plotting in R. It is impossible to give a general overview during this course. There are many basic plot formats, plots in 3D, maps. Some packages like *lattice, ggplot...* contain only graphical functions that are widely used. In the web are many galleries with examples of R plots. You could check, for instance, the following ones:
http://scs.math.yorku.ca/index.php/R_Graphs_Gallery
http://zoonek2.free.fr/UNIX/48_R/03.html
http://en.wikibooks.org/wiki/R_Programming/Graphics

If we wanted to export the plot with a specific format **(file Rcourse.7)**

The only thing we need is to use a function for the desired format to export, before the code of the plot, and, after the code of the plot, close the output with the *dev.off()*

function. For the plot above:

```
pdf("Myplot1.pdf")
par(oma=c(2,4,2,11),mar=c(5.1,4.1,6,2.1),xpd=TRUE)
plot(tm,N2[,"Brach"],pch=19,xlab="Time (days)",ylab="",ylim=c(0,20))
error.bar(tm,N2[,"Brach"],N2[,"SdBrach"])
spl1<-spline(tm,N2[,"Brach"],n=20*length(N2))
points(spl1$x,spl1$y,type="l",lwd=2,col="red")
mtext(expression(paste("Rotifers (ind mL"^"-1",")")),2,2,at=10)
par(new=TRUE)
plot(tm,N2[,"Scen"],pch=19,xlab="",ylab="",yaxt="n")
error.bar(tm,N2[,"Scen"],N2[,"SdScen"])
spl2<-spline(tm,N2[,"Scen"],n=20*length(N2))
points(spl2$x,spl2$y,type="l",lty=2,lwd=2,col="green")
axis(2,line=4)
mtext(expression(paste("Scenedesmus (cells ",10^6,"mL"^"-1",")")),2,6,at=0.25)
par(new=TRUE)
spl3<-spline(tm,N2[,"Eggs"],n=20*length(N2))
plot(spl3$x,spl3$y,type="l",lty=3,lwd=1,xlab="",ylab="",yaxt="n",ylim=c(0,0.35))
axis(4)
mtext(expression(paste("Eggs (ind"^"-1",")")),4,3,at=0.175)
par(new=TRUE)
spl4<-spline(tm,N2[,"Def"],n=20*length(N2))
plot(spl4$x,spl4$y,type="l",lty=4,lwd=1,xlab="",ylab="",yaxt="n",ylim=c(0,6))
axis(4,line=5)
mtext(expression(paste("Level of defense (average cells aggregate"^"-1",")")),4,7,at=3)
par(new=TRUE)
spl5<-spline(tm,D[,"N2"],n=20*length(D))
plot(spl5$x,spl5$y,type="l",lty=5,lwd=1,xlab="",ylab="",xaxt="n",yaxt="n",ylim=c(0,2))
axis(4,line=9)
mtext(expression(paste("Dilution rate","(day"^"-1",")")),4,11,at=1)
legend(x=17,y=2.7,c("Scenedesmus","Brachionus","Eggs","Defense","Dilution"),lty=c(2,1,3,4
 ,5),lwd=c(2,2,1,1,1),col=c("green","red","black","black","black"),bty="n")
dev.off()
```

Now, the pdf file “Myplot1” should appear in your working directory. Other format options are those of jpeg(), tiff(), png(), bmp(). You can play with the size and resolution of the plots. Just check the help for these functions.

Other graphic libraries

As mentioned earlier, the possibilities for plotting with R are almost nevere ending. Here, I show a few example of other graphic libraries that contain demos (the lattice library is installed by default, but not loaded, the others, you need to install them):

```
> require(lattice)
> demo(lattice)

> require(plotrix)
> demo(plotrix)

> require(rgl)
> demo(rgl)
```

Bibliography

For a general introduction to R:

- *The R book*. Michael J. Crawley. John Wiley & Sons. 2012.

- Beginning R. M. Gardener. Wiley & Sons, 2012.

- *The art of R programming: A tour of statistical software design*. Norman Matloff. No Starch Press. 2011.

- Learning R. Richard Cotton. O'Reilly Media. 2013.

- R in a nutshell: A Desktop quick reference J.R. Adler. O'Reilly Media. 2012.

For more advanced programming:

- Programming with R. J.M. Chanbers. Springer 2008.

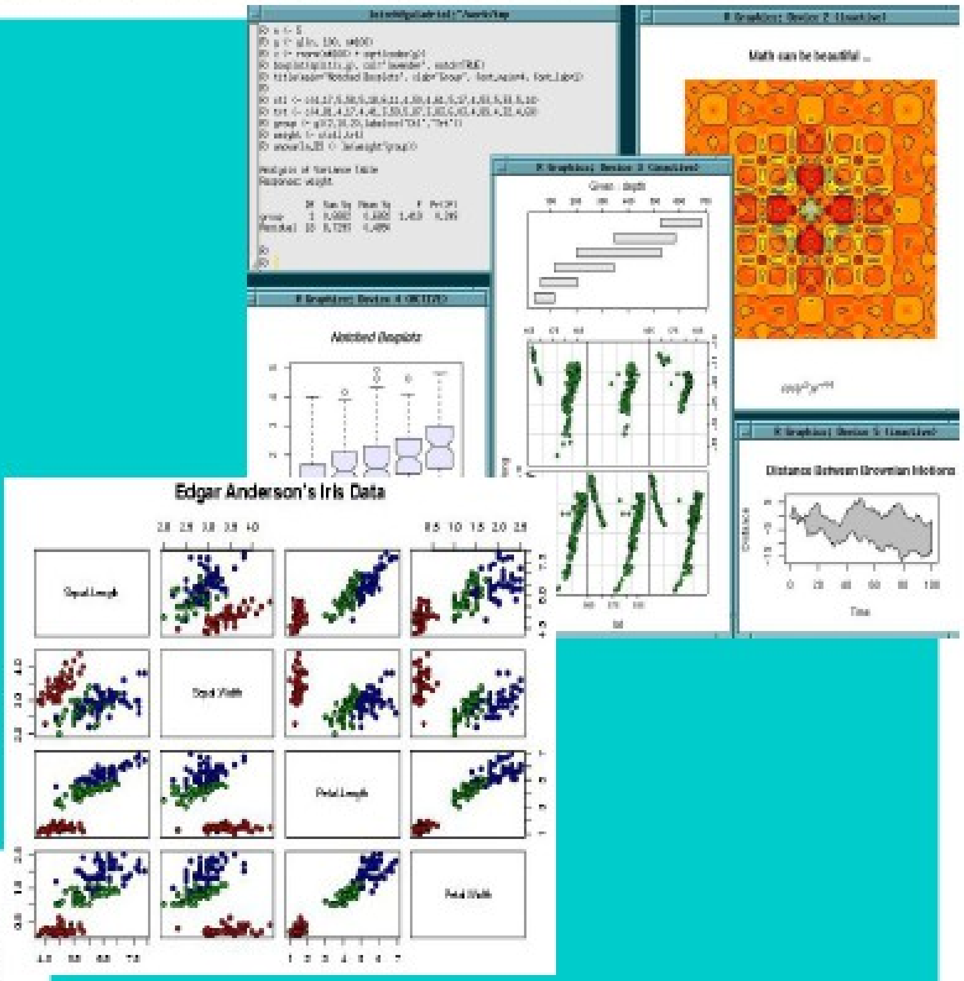
- Guide to programming and algorithms using R. Özgür Ergül. Springer 2013.

For graphics:

- *R Graphics*. Paul Murrell. Chapman and Hall, 2005.

- *Gráficos estadísticos y mapas con R*. Cástor Guisande, Antonio Vaamonde Liste. Díaz de Santos. 2013.

INTRODUCTORY STATISTICS AND PROGRAMMING WITH *R* FOR BIOLOGICAL SCIENCES


Math can be beautiful ...
Notched Boxplots
Edgar Anderson's Iris Data
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
Distance Between Brownian Motions
Time

Module II: Basic Statistics

Aldo Barreiro Felpeto, June 2015

Table of contents

Section 1: Descriptive statistics and regression

Contents

1.1 Descriptive statistics and distributions

Descriptive statistics (file R.S1.1)

Descriptive statistics are summary values, mainly of univariate or bivariate data, that provide information about the central tendency, dispersion, range of variation, the relationship between variables, etc.

Basic measures of central tendency

We may start with the mode, which is the most frequent value of a variable. There is no specific function for the mode in R default packages. We can easily program it, but, in this course we will use the function mlv() from the package *modeest*. This function has different methods for the estimate of the mode. In this example, we are using the simplest one, which is returning the most frequent value from a variable (method “mfv”, which stands for “most frequent value”).

```
> require(modeest)
> x<-c(3,6,7,4,3,2,5,4,3,3,7,8,3)
> mlv(x,method="mfv")
Mode (most likely value): 3
Bickel's modal skewness: 0.4615385
Call: mlv.default(x = x, method = "mfv")
```

The mode is not always a good measure of central tendency, since it could be positioned far from the majority of data. In addition, sometimes, the data have more than one mode.

The median is the value in the “middle”. It has the same number of values to its “left” and to its “right”. There is a function in the *stats* package. By default, it does not calculate the median if there are NAs in your data:

```
> y<-c(3,6,NA,7,4,3,2,5,4,3,NA,3,7,8,3)
> median(y)
[1] NA
> median(y,na.rm=TRUE)
[1] 4
```

it is very informative as a measure of central tendency, since it is not much affected by outliers, as it is the mean.

The mean is one of the more used descriptive measures of data. It has several definitions in mathematics or statistics, depending on the context. It could be defined as the central value of a series of values, when summed up and divided by the total number of values, or the expected value, based in a distribution of probability. Its problem, as a measure of central tendency, is that it could be severely affected by outliers, usually whit small data sets. This is the arithmetic mean in R:

```
> mean(y,na.rm=TRUE)
[1] 4.461538
```

A sometimes useful argument of this function is “trim”, which indicates the fraction (from 0 to 0.5) of observations to be trimmed from each end of the variable. This could be useful to eliminate outliers. The mean would be then called “truncated mean”.
There are other kinds of mean, among them the geometric mean and the harmonic mean. Their usefulness depends on the kind of data. The geometric mean is useful for those variables that are interpreted according to their product, not their sum. For instance, a growth rate. This is the formula:

$$G=\sqrt[n]{x_1 * x_2 * \ldots * x_n}$$

As an example, imagine a microorganism that growths at a daily rate of 0.7 during one day of your experiment. Imagine that during the second day it would grow at a rate of 0.2. So the mean of the growth during these two days is not interpretable as (0.7 + 0.2) / 2 = 0.45. Rather, we may use the geometric mean, which is the *ith* root of the product of *i* values, which is the same as the product of *i* values rose to the 1/i power. In this specific example: (0.7 x 0.2)^(1/2) = 0.37. In R, there is no built – in function to compute this mean in the default packages. It is, obviously, very easy to program as a user defined function. But here, we are going to use the built – in function from package *psych*:

```
> require(psych)
> geometric.mean(y)
[1] 4.096351
```

The harmonic mean is useful for the calculation of averages of rates. It is the inverse average of the inverted values:

$$H=\frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+\ldots+\frac{1}{x_n}}$$

It is also very easy to program, but there is a built – in function as well in the *psych* package:

```
> require(psych)
> z<-c(0.6,0.5,0.6,1.12,0.2,0.4,0.7)
> harmonic.mean(z)
[1] 0.461901
```

Basic measures of dispersion

The variance expresses the spread of the data with respect to the mean. It is the sum of the squared differences between each value and the mean. There is a very simple function to compute it in the default package *stats*:

```
> var(z)
[1] 0.08158095
```

The problem of the variance is that it is squared with respect to the actual values of the variable. If we want to use a measure of dispersion in the same units as the original variable, we will need to use the standard deviation, which is the squared root of the variance. The R function is:

```
> sd(z)
[1] 0.2856238
```

Sometimes, the standard deviation is computed averaged by n – 1 data, in order to reduce the bias. But the R function does not use this feature.
Another measure of dispersion, the coefficient of variation, is useful when we want to compare variables measured in different scales. It is the standard deviation normalized by the mean. It is very easy to calculate simply dividing the standard deviation by the mean. But there is a function as well, in the R package *raster*:

```
> require(raster)
> cv(x)
[1] 43.51527
> cv(z)
[1] 48.52831
```

The range is the whole range of distribution of our data. So the output from the function are the minimum and maximum of our variable:

```
> range(z)
[1] 0.20 1.12
```

Another interesting measure of dispersion is the interquartile range, which is the difference between the first and the third quartile. So we are excluding the variable values that are smaller than the first quartile (the smaller 25% of the data) and greater than the third quartile (the largest 25% of data). The R function is IQR():

```
> a<-rnorm(1000)
> IQR(a)
[1] 1.380011
```

On page 31 from the pdf of week I (script *Rcourse.4*) there is an example of a function, stat.desc(), from package “pastecs”, that returns at once the most common measures of descriptive statistics.

Correlation coefficients

The correlation coefficients are a measure of the strength and direction of the association or dependence (usually linear) between two variables. It is always important to remind that statistic correlation does not mean causation.
We may differentiate between two main groups: parametric coefficients and non – parametric coefficients. As we will show later in this course, we use parametric methods when it is possible to assume that our data follow certain statistical distribution, often the Normal distribution. Any distribution is defined by a mathematical function with parameters. The non – parametric methods are used when we can not assume that our data follow a

specific distribution, either because they do not significantly fit to any distribution or because we have so few data that it is not reliable to test the assumption.
So, the parametric correlation coefficients are used when our data follow a Normal distribution. The most famous one is the Pearson's coefficient (*r*). It measures the strength of the linear dependence among two variables. It is defined as the covariance between two variables divided by the product of their standard deviations:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

Its range of possible values goes from -1 to 1. The sign is positive when the relationship is positive and negative when it is negative. The closer the absolute value to 1 the stronger the relationship. Relationship here means statistical dependence. A value of 0 means no relationship, so both variables would be independent. There are statistical tests that calculate the significance of this coefficient, but we are not going to take care of this details now.
Among the most important non – parametric coefficients are the Spearman's coefficient (ρ) and Kendall's coefficient (τ). Both are calculated based in the ranks of the paired data of the two variables. The closer the rank of two paired values, the stronger the correlation. Their range of possible values is also between -1 and 1, and the interpretation the same as with Pearson's coefficient. There also exist statistical tests for the estimate of their significance, but we are going to skip this point now. There are two different kinds of Kendall τ, named *a* and *b*. Their use depends on the existence of ties in the ranks, but we are not going to get into this now. Kendall coefficient usually performs better for ordinal variables.
The main difference between these non – parametric coefficients and Pearson's coefficient is that that the relationship between the variables does not have to be necessarily linear. It is only assumed to be monotonic. We are going to see an example of this at the same time that we calculate these coefficients with R.
We are going to create two kinds of relationships between three variables, one is going to be linear and the other non – linear, but monotonic, and see what these coefficients tell us. First, we create 30 random numbers, withdrawn from a normal distribution with average 0 and standard deviation 1. These are going to be our x variable:

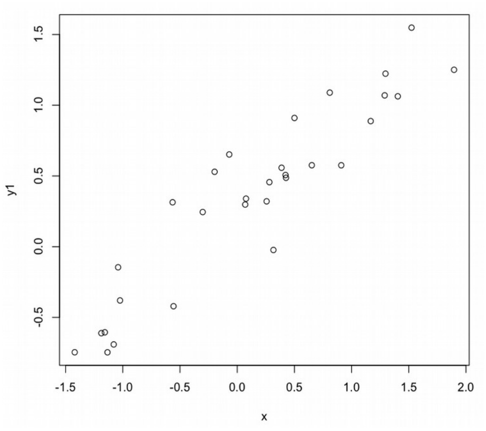
```
> x<-rnorm(30)
```

Now, we create another variable, y1, that is going to be linearly related with x, but adding some noise:

```
> y1<-(0.7*x+0.3)+rnorm(30,0,0.25)
```

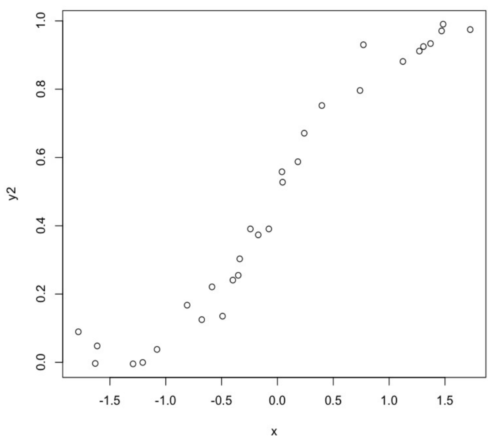
As you have seen, in order to create y1, we used a linear function applied to the 30 values of x and then added some noise from a normal distribution with average 0 and standard deviation 0.25. If we plot x against y1, this is how the relationship looks like:

```
> plot(x,y1)
```



And now, in the same way, we are going to create a new variable y2, that is non linearly related wit x. So we will use a sigmoid function, the logistic, and add some noise as well:

```
> y2<-(1/(1+exp(-2.5*x)))+rnorm(30,0,0.05)
> plot(x,y2)
```



And now, we are going to calculate the correlation coefficients described above for these two relationships. In R, we have the function cor.test() in the default package *stats*, that can calculate all the three coefficients, just by modifying the value of the argument “method”. For the method “kendall”, this R function already adjusts the calculations depending on the existence of ties in the ranks.
These are the coefficients between x and y1, which “skeleton” is a linear relationship:

```
> cor.test(x,y1,method="pearson")

	Pearson's product-moment correlation

data:  x and y1
t = 13.513, df = 28, p-value = 8.615e-14
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8590753 0.9670205
sample estimates:
      cor
0.9311539

> cor.test(x,y1,method="spearman")

     Spearman's rank correlation rho

data:  x and y1
S = 266, p-value = 1.806e-08
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.9408231

> cor.test(x,y1,method="kendall")

     Kendall's rank correlation tau

data:  x and y1
T = 392, p-value = 3.173e-13
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.8022989
```

And these are the coefficients between x and y2, which “skeleton” is a sigmoidal relationship:

```
> cor.test(x,y2,method="pearson")

     Pearson's product-moment correlation

data:  x and y2
t = 19.8152, df = 28, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9293590 0.9839337
sample estimates:
      cor
0.9661443

> cor.test(x,y2,method="spearman")

     Spearman's rank correlation rho

data:  x and y2
S = 74, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.9835373

> cor.test(x,y2,method="kendall")

     Kendall's rank correlation tau
```

```
data:  x and y2
T = 417, p-value = 1.11e-15
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.9172414
```

So, as we can see, the spearman coefficient always finds the relationship stronger, by little in the case of the linear relationship. But in the sigmoidal relationship, Spearman's definitely indicates stronger relationship than Pearson's coefficient. The value of Pearson's coefficient is still strong for the sigmoidal relationship, because the slope is sharp (so along most of its range, the relationship is linear) and the noise is very low. Kendall's coefficient always indicates the lower strength of correlation. As we mentioned earlier, this coefficient is more suitable for ordinal variables.

*There exist as well measures of dependence for qualitative variables and tests for this dependence, but due to our limited time budget, I am going to mention them briefly: The Φ (phi) coefficient is the main measure of dependence for qualitative variables. In R there is the function phi() in package *psych*. Its multivariate version is Cramer's *V* coefficient (R function cramersV() from package *lsr*). Tests for independence can be the Chi – square test for polytomous variables (chisq.test() function) and the Yates test for dichotomous variables (chisq.test() function with argument “correct” set to TRUE). All these measures and tests with qualitative variables are calculated from their frequencies in contingency tables (we will see them in session 3).

Basic features of working with distributions **(file R.S1.2)**

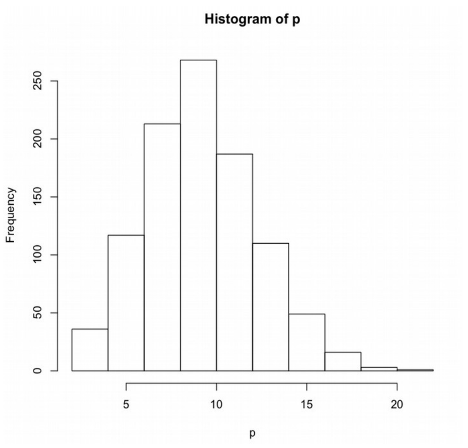
The distribution of a variable, in statistics, is often used in the context of the frequency of its values, or the probability of obtaining an outcome from a specific experiment. An outcome is a specific value of some variable, defined from a probabilistic point of view.
We will use the features of R related with distributions, mainly, for these purposes: estimating the goodness of fit of data to specific distributions, obtaining the probability of a value under the assumption of certain probability distribution, and the generation of random numbers.
Each distribution has its own features: Some of them could be specific for discrete or continuous variables. They may have only positive values, negative or both. They can be symmetric or skewed, have different degrees of kurtosis, etc. Each distribution is defined by a different mathematical function, with its specific parameters.

Generating random numbers

We have already seen how to generate random numbers drawn from the Normal (or Gaussian) distribution with the function rnorm(). The parameters of the Normal distribution function are the mean and the standard deviation. The default values that rnorm() takes for these parameters are, respectively, 0 and 1.
In R, there are equivalent functions for many other statistical distributions, a few ones that are commonly used are: rbinom() (binomial), rchisq() (chi – squared), rgamma() (Gamma), rlnorm() (log – normal), rpois() (Poisson) and many others...

Let's see an example of random number generation from a Poisson distribution:
The Poisson distribution function has one single parameter, called lambda (λ), which is equal to both its mean and variance. It is a discrete distribution, usually for count data. Its values have to be always positive.

```
> p<-rpois(1000,lambda=10)
> hist(p)
```



Calculating probabilities under specific distributions

Another application that we might want to use is to calculate the probability of a specific value under the assumption of certain probability distribution. This is for instance, what we do in a typical hypothesis test. We obtain the value of a contrast statistic (Fischer's *F*, Student's *t*, etc.) and we calculate its probability assuming that distribution. These calculations are often integrated within the code of built – in functions of statistical tests, so we obtain the probability value already from the function output. But, in some cases, we might need to calculate the probability directly, because we do not have a built – in function that applies a specific test, and then, we have to do it "by hand".
As an example, there is a common application of the likelihood ratio test which consist in testing the significance of the difference between the likelihood of two fitted statistical models (forget now about what fitting models and likelihood mean, we will talk about it in the forthcoming sections). For models fitted with built – in functions like lm(), glm(), and others, we can apply the likelihood ratio test directly with the function anova(), and obtain a probability value from the output. But, in some cases, we might code models with our own user – defined functions. In such a case, the function anova() is not ready to "recognize" the model object, and hence, we might need to calculate our likelihood, and do the test by hand. For instance, imagine that we are comparing two models, "model 1" with a likelihood value of -13275 and "model 2" with a value of -11923. The one with greater likelihood (model 2) is supposed to be better, but because it has one more parameter, we would prefer the simpler "model 1" if this difference in likelihood is not significant. In order to know it, we have to apply a likelihood ratio test, which consist in calculating the probability of the test statistic called *D*, under a chi-squared distribution. So, in this case, in R, *D* is calculated as follows (do not think about the details of the formula now):

```
> D<--2*log(-13275.43/-11923)
> D
```

```
[1] -0.2148913
```

Now that we have the value of *D*, we calculate its chi – squared probability with 1 degree of freedom. We have set the argument “lower.tail=TRUE” because the difference between the likelihood values could be negative or positive. It would be significant either a highly positive value of *D* or a highly negative one.

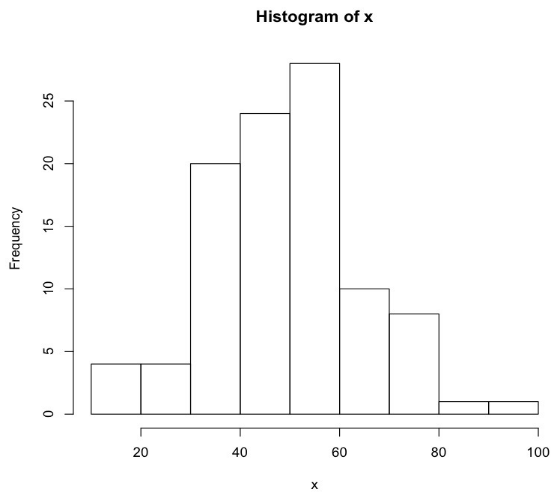
```
> pchisq(D,1,lower.tail=T)
[1] 0
```

The probability is lower than 0.05, so we consider the difference significant, so in this case we would choose model 2.
Obviously, there are functions equivalent to pchisq() for many other statistical distributions.

Fitting data to distributions and goodness of fit

In order to fit our data to a specific distribution, we simply have to estimate the parameters of the that distribution function from our data.
A goodness of fit test would contrast the empirical frequencies with the theoretical ones. There are different tests that we can apply. For instance, we have the Shapiro – Wilks test that is only used for the test of Normality. But there is also the Kolmogorov – Smirnov test, that could be used for a Normal distribution, or also other kinds of distribution.
We are going to see a single example of these things. Estimate the parameter values needed to define a Gamma distribution and then perform a goodness of fit test for this same distribution. We are going to use the artificial data from a normal distribution. The Gamma distribution is continuous and has always positive values.

```
> rnorm(100,50,15)
> hist(x)
```



This is the real, empiric distribution of our data. Now, if we want to fit them to a Gamma distribution, we need to estimate the two parameters of the Gamma function, the “shape” parameter, and the “rate” parameter.
There are many ways to estimate this parameters in R. For now, we are going to choose the simplest, the most convenient. This consists in using the function fitdistr(), from the default package *MASS*. This function estimates the parameters by maximum likelihood for

univariate distributions. First, we have to load the package *MASS*, because, although it is installed by default, it is not loaded by default.

```
> require(MASS)
> fitdistr(x,"gamma")
     shape         rate    
  8.97730117   0.18328807 
 (1.24648000) (0.02617341)
Mensajes de aviso perdidos
1: In densfun(x, parm[1], parm[2], ...) : Se han producido NaNs
2: In densfun(x, parm[1], parm[2], ...) : Se han producido NaNs
```

The values provided are the estimates, and within parenthesis, the standard error of these estimates. The production of NaNs during the calculations does not seem to be good... Now, we are going to apply a goodness of fit test in order to check if the empirical frequencies meet the theoretical frequencies for this kind of distribution. If so, our parameter estimates would make sense. For this purpose, we will use the Kolmogorov – Smirnov test with the method for Gamma probability. The null hypothesis is that the data actually fit the gamma distribution:

```
> ks.test(x,"pgamma",8.98,0.183)

     One-sample Kolmogorov-Smirnov test

data:  x 
D = 0.0808, p-value = 0.5313
alternative hypothesis: two-sided
```

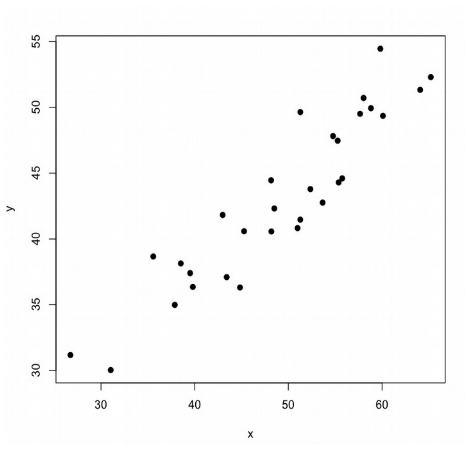
The probability value (p – value) is greater than 0.05, so there is no evidence to reject the null hypothesis, and so we are allowed to say that these data fit a Gamma distribution. It might appear curious that data drawn from a Normal distribution fit another different distribution. But this is possible, since statistical tests always deal with probabilities, and, additionally, there are not perfect. In this case, it might happen that the two distributions are not that different, and the sample size is a bit small.

1.2 Linear regression (file R.S1.3)

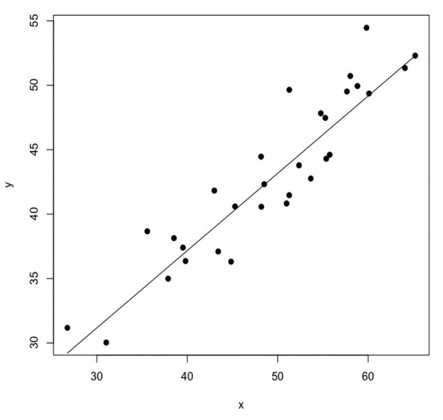
Introduction

Fitting data to a linear function is one of the most frequently performed tasks in statistics. It is the basis of simple and multiple linear regression and the linear model, that we will see in the forthcoming sections. We have an independent variable, or predictor, usually denoted as *x*, and a dependent variable or response variable, usually denoted as *y*. The basic functional form of a linear model is $y = ax + b$ and fitting this model to our data consist in finding the best estimates for the two parameters from this model: the intercept *b* and the slope *a*. This estimates could be performed with different methods. But in this case we are going to use only the most common one: minimization of residual squared sum. Here is a short explanation of this method:
First, we have to explain what the residuals are. Imagine we have an empiric relationship

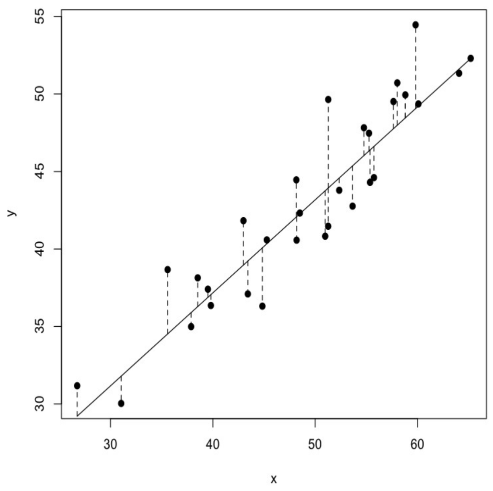
between two variables, like this one:



If we fit a linear function to this relationship, we will have theoretical values of the *y* variable, constituting a straight line. These theoretical values of *y* correspond to each real value of *x*. In this way:



Now, the differences between the real values of *y* and the theoretical or fitted values of *y* are called the **residuals** of the model. These are represented in this plot by the discontinuous lines:

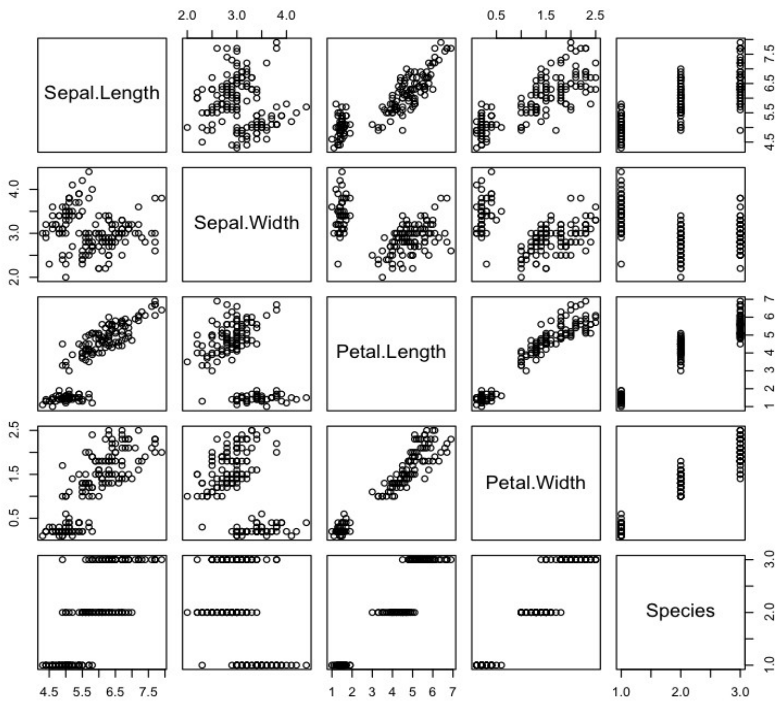


So the method of minimization of residuals squared sum consist in finding parameter estimates that minimize the value of the sum of the squared values of the residuals. The squares are applied because we need a positive value, and some of the residual values will be negative (those of points laying below the line). The minimum value of this sum is found by iterating a matrix algorithm that calculates the residuals for different fitted values of *y* calculated from iterated values of the two parameters of the linear function (*a* and *b*). An important property of the residuals is that their sum is always equal to 0.

Example of fit

Now, let's proceed with an example of a linear fit. This technique is often called "linear regression", because one of the first data sets that was applied to develop its foundations was the height of adult humans across generations. The parents had smaller heights than their offspring, so the fitted line showed the "regressive" way of human height from the present to the past.
For our example, we are going to use the data from *iris*, that offers the possibility of comparing different morphological measures of those flowers.
First, we load the data and we can have a look at all the possible pair – wise combinations of the variables. This can be done simply with the plot() function applied to the whole data frame:

```
> data(iris)
> plot(iris)
```



This plot is a very good summary plot for data exploration. Note the symmetry with respect to the diagonal. Thinking about which variables we could use for our example, it looks like only petal length versus petal width follows the same pattern for all the three species. And it looks pretty linear.

So, we start defining the model. The main function in R for a linear model is lm(). The first argument of this function is the model formula. In this formula you only have to indicate the dependent and the independent variable. The argument *data* is useful in order to avoid writing the “$” to locate the variables:

```
> model<-lm(Petal.Width~Petal.Length,data=iris)
```

So the way the formula is written, it will be interpreted by R as “Petal width predicted by petal length”. With the function summary() we can access to the main results of the model fit:

```
> summary(model)

Call:
lm(formula = Petal.Width ~ Petal.Length, data = iris)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.56515 -0.12358 -0.01898  0.13288  0.64272 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -0.363076   0.039762  -9.131  4.7e-16 ***
Petal.Length  0.415755   0.009582  43.387  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2065 on 148 degrees of freedom
Multiple R-squared: 0.9271,    Adjusted R-squared: 0.9266 
F-statistic:  1882 on 1 and 148 DF,  p-value: < 2.2e-16
```

Interpretation of this output

So this summary function shows us several things that are stored in the model of the object that we called “model”. These things are shown because they are considered the most important information. First, there is a reminder of the formula of the model. Then, some descriptive statistics about the model residuals (minimum, maximum and quartiles). Below, the better estimates for the coefficients of the linear function, as we know, calculated by minimization of the sum of squared residuals. Besides the estimate, it is also shown the standard error of the estimate, and the statistical significance of the estimates with the Student's *t* as contrast statistic. At the end are shown the residual standard error, with the degrees of freedom for the error of the model. This residual standard error is the value that is minimized by iteration, as we explained above. But, do not be mistaken, it is not the sum of the residuals, that, as we mentioned above, is always = 0. This residual standard error, is interpreted as the unexplained variance. Because it is calculated as the sum of all the squared differences between a real value and the expected theoretical value. So if you remember how the variance is calculated, this is strongly related.
Do you know what degrees of freedom mean? They have interpretations expressed in a different way, depending if we are in a context of physics or statistics, but in essence, they are equivalent. They are the number of independent dimensions that are available in order to estimate a parameter. The residuals are an estimate of the error of the model. And, as

we said above, they sum is equal to 0. In this example, we have 150 data. So we will have 150 residuals. Imagine if we knew the values of 149 residuals. Then, we necessarily know as well the value of the one that is missing, because their total sum must be 0. This means that we have only 149 “free” dimensions. 149 residuals that could take any value, but an additional one is constrained to make the total sum equal to 0.
But the degrees of freedom for the error are 148, not 149. This is because what we have explained above applies to the degrees of freedom of an error estimate. But we are in a slightly more complicated case. Here we are estimating, besides the error, two model parameters (*a* and *b*, the slope and the intercept). Then, the other reduction in degrees of freedom comes from the estimation of the fitted or theoretical values of the dependent variable. For this calculation, we need the values of the independent variable and two parameters. The values of the independent variable are known, they are empiric. Then, the value of one parameter is also constrained by the value of the other parameter. The constrain is related with the fitted values of the dependent variable. Because, in this situation, we know the values of the residuals and the values of the independent variable, the fitted values of the dependent variable have to be in accordance with those previous values. And for its calculation we have only two elements remaining: the two parameters. And, when we know the value of one of them, the value of the other is constrained to yield the necessary values of the fitted dependent variable.
So we could say that we have 149 (150 data – 1) degrees of freedom for the error and 1 degree of freedom for the model (2 parameters – 1). The total degrees of freedom are hence 148.
The last things that this summary provides are the *R*2 coefficient of determination and its statistical significance estimated from the *F* statistic. This coefficient, for models in general, indicates how well the data fit the model, as a sort of the proportion of variance that is explained by the model. But this coefficient is not the same as the squared value of the Pearson's correlation coefficient. This is its most general formula:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

The numerator is the residual sum of squares, so the varaince due to each individual data, not explained by the model. The denominator is the total variance. The difference between these two quantities is the variance explained by the linear regression model. So, if the model explains a large amount of variance, the whole quotient has a low value. And if this value is substracted to 1, the coefficient would have a value close to 1.
The summary shows two values of this coefficient. The “adjusted” value is a correction performed in order to take into account the number of parameters, related with the number of explanatory variables included in the model. In our case we have only one explanatory variable, our independent variable. But if we had more (like in a multiple regression) each explanatory variable adds one extra parameter to the model. In the case of the multiple regression, this parameter is a slope. If we add any new explanatory variable to a model, the chances to get a better predictive model increase, the unexplained variance is reduced and the value of the *R*2 increases as well. So this “adjustment” penalizes the number of explanatory variables in a model.

Interpretation of the coefficients and its significance

The interpretation of the intercept is straight forward: the expected intercept of the predicted line with the OY axis. The interpretation of the slope is very easy. It tells us how many “units” changes the dependent variable with respect to an increase in 1 unit of the independent variable. In this case, the slope is 0.41. This means that with the increase in one unit of petal length generates an increase of 0.41 units of petal width. If the relationship was negative, instead of increase would be a decrease.
The significance of these coefficients is calculated with a Students's *t* statistic. The significance of the slope is strongly related with the confidence interval defined as [Estimate – standard error, Estimate + standard error]. If this interval contains the 0, or is close to it, it is very likely that the slope is not significant. Because if the value 0 is included in this confidence interval, it means that 0 is a highly probable value for the slope. And a slope with 0 values means no effect of the independent variable on the dependent variable. In our case, the slope is highly significant. In a similar way, the intercept is usually significant the farther it is from 0 (obviously, if it was equal to 0, we do not need a model with intercept).

Extracting more components from the model object

The object we called “model” has more components than those shown with the function summary(). We can see which objects are those with the function objects():

```
> objects(model)
 [1] "assign"        "call"          "coefficients" "df.residual"   "effects"
 [6] "fitted.values" "model"         "qr"           "rank"          "residuals"
[11] "terms"         "xlevels"
```

We can access each of these components individually. For instance, if we wanted to see the fitted values:

```
> model$fitted.values
         1          2          3          4          5          6          7          8 
0.21898206 0.21898206 0.17740652 0.26055760 0.21898206 0.34370869 0.21898206 0.26055760 
         9         10         11         12         13         14         15         16 
0.21898206 0.26055760 0.26055760 0.30213314 0.21898206 0.09425544 0.13583098 0.26055760 
        17         18         19         20         21         22         23         24 
0.17740652 0.21898206 0.34370869 0.26055760 0.34370869 0.26055760 0.05267990 0.34370869 
        25         26         27         28         29         30         31         32 
0.42685977 0.30213314 0.30213314 0.26055760 0.21898206 0.30213314 0.30213314 0.26055760 
        33         34         35         36         37         38         39         40 
0.26055760 0.21898206 0.26055760 0.13583098 0.17740652 0.21898206 0.17740652 0.26055760 
        41         42         43         44         45         46         47         48 
0.17740652 0.17740652 0.17740652 0.30213314 0.42685977 0.21898206 0.30213314 0.21898206 
        49         50         51         52         53         54         55         56 
0.26055760 0.21898206 1.59097494 1.50782385 1.67412602 1.29994614 1.54939939 1.50782385 
        57         58         59         60         61         62         63         64 
1.59097494 1.00891735 1.54939939 1.25837060 1.09206844 1.38309723 1.29994614 1.59097494 
        65         66         67         68         69         70         71         72 
1.13364398 1.46624831 1.50782385 1.34152169 1.50782385 1.25837060 1.63255048 1.29994614 
        73         74         75         76         77         78         79         80 
1.67412602 1.59097494 1.42467277 1.46624831 1.63255048 1.71570156 1.50782385 1.09206844 
        81         82         83         84         85         86         87         88 
```

```
1.21679506 1.17521952 1.25837060 1.75727710 1.50782385 1.50782385 1.59097494 1.46624831
        89         90         91         92         93         94         95         96
1.34152169 1.29994614 1.46624831 1.54939939 1.29994614 1.00891735 1.38309723 1.38309723
        97         98         99        100        101        102        103        104
1.38309723 1.42467277 0.88419073 1.34152169 2.13145698 1.75727710 2.08988144 1.96515481
       105        106        107        108        109        110        111        112
2.04830589 2.38091023 1.50782385 2.25618360 2.04830589 2.17303252 1.75727710 1.84042819
       113        114        115        116        117        118        119        120
1.92357927 1.71570156 1.75727710 1.84042819 1.92357927 2.42248577 2.50563685 1.71570156
       121        122        123        124        125        126        127        128
2.00673035 1.67412602 2.42248577 1.67412602 2.00673035 2.13145698 1.63255048 1.67412602
       129        130        131        132        133        134        135        136
1.96515481 2.04830589 2.17303252 2.29775914 1.96515481 1.75727710 1.96515481 2.17303252
       137        138        139        140        141        142        143        144
1.96515481 1.92357927 1.63255048 1.88200373 1.96515481 1.75727710 1.75727710 2.08988144
       145        146        147        148        149        150
2.00673035 1.79885264 1.71570156 1.79885264 1.88200373 1.75727710
```

Assumptions of linear regression, and how to test them

This part should be at the beginning, but we left it at the end for convenience. It is easier to explain when we know well what the residuals are, and how to extract them. The point is that a linear model as we presented it here can not be estimated for any kind of data. It is only possible if our data meet certain assumptions. These assumptions are valid as well for multiple linear regression, that we will talk about next in this section:

- Linearity: the dependent or response variable has to be a linear combination of the parameters. In simple linear regression, we are testing this assumption directly with the model itself. But in multiple regression, it is often good to perform a simple regression with each independent or predictor variable. Those that do not show a significant coefficient of determination should be excluded.

- Normality of residuals: residuals must follow a Normal distribution.

- Homoscedasticity: the variance of the model residuals, has to be homogenous across all the range of the predicted values.

- Independence of residuals: Residuals should be uncorrelated with each other. Usually these last two assumptions are not met by our data when there is some relevant predictor variable that is missing in the model.

- Predictor variables should be linearly independent: obviously, this only applies to multiple linear regression. This can be simply tested by performing simple linear regressions with all possible pair – wise combinations of the predictor variables.

Statisticians usually combine two approaches to test the above assumptions. There are specific tests for some of them, but they also may decide based in specific plots of model components. The best way is probably combining both. Here, we are going to explain first the tests, and then the plots.
First, the Normality of residuals could be tested with the Shapiro – Wilks test or the

Kolmogorov – Smirnov test. The last one is better for large amounts of data, so here, we are going to see an example with the Shapiro – Wilks test. First, we have to create an object containing the residuals. We can access the residuals, as we showed above like this: model$residuals. But is perhaps more convenient to use a specific function, which is residuals(). Then, we apply the test:

```
> res<-residuals(model)
> shapiro.test(res)

     Shapiro-Wilk normality test

data:  res
W = 0.9838, p-value = 0.07504
```

The null hypothesis is that there is no difference between a Normal distribution and the distribution of the residuals. Because the p – value is greater than 0.05, there is no evidence to reject the null hypothesis. So we are allowed to believe in the normality of these residuals.
The homoscedasticity of residuals is tested with the non constant error variance test, from the *car* package:

```
> require(car)
> ncvTest(model)
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 37.74426    Df = 1     p = 8.065366e-10
```

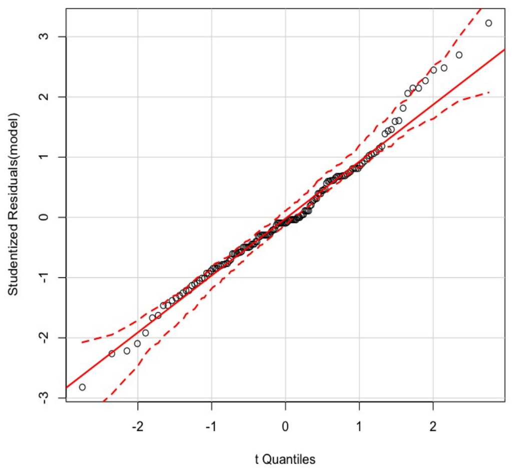
The null hypothesis is that the errors are homogeneous, but the p – value is less than 0.05, so the null hypothesis should be rejected. Our errors are not homogeneous.

The independence of residuals could be tested with the Durbin – Watson test, also in *car* package:

```
> durbinWatsonTest(model)
 lag Autocorrelation D-W Statistic p-value
   1       0.2723717       1.45491   0.002
 Alternative hypothesis: rho != 0
```

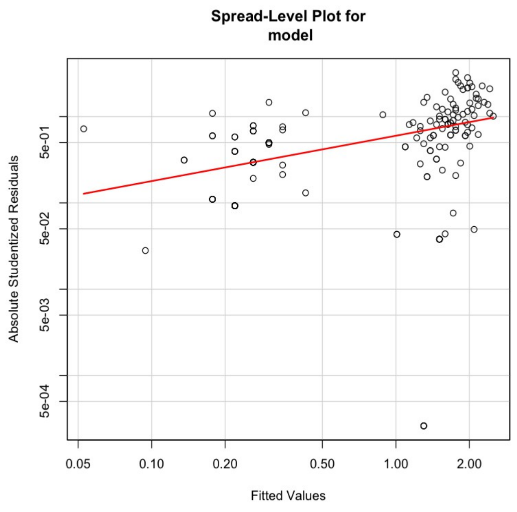
The null hypothesis is that the residuals are indeed uncorrelated, but the p – value is significant, so the null hypothesis should be rejected, the residuals are autocorrelated.
Now, lest's see a few plots that can help us to interpret how our data meet or not these previous assumptions. For the Normality, we have the QQ plot, which shows the theoretical distribution of quantiles of the Normal distribution in a straight line. If the residuals fit well this line, it means that they are probably Normal. The function qqPlot() is also in the package *car*:

```
> qqPlot(model)
```



Our residuals are a bit sigmoidal but in general look very good. So the result of the test is supported by this plot.
In order to check graphically the homoscedasticity of the residuals, we can plot the fitted values of the dependent variable versus the residuals. If everything is right, the residuals should be in average at the same distance from the fitted values across all the range of fitted values. We can directly plot the residuals and fitted values extracting them from the model, but the car package contains a function that performs this task directly:

```
> spreadLevelPlot(model)

Suggested power transformation:  0.4739141
```



As you can see, the residuals are clearly not homogenous, in agreement with the previous test. Actually, the plot function additionally suggest a power transformation of our data in order to get residual homogeneity. If we look at the red line showing a fitted regression, we can see that there is a certain positive relationship. This is not what we should expect.

Because the residuals are normalized (with a Student method) their regression against the fitted values should be a flat line.
The function plot() can also show similar plots to those obtained with those functions from the package *car* showed above, as well as some additional ones, and it works with the same argument (the name of the object that stores the model). It is a bit more convenient because it does not need to install or load any package in order to be used.

What can we do when the assumptions are not met

So these data are not suitable for a linear regression. We can not trust our model, despite the significance of the coefficients. The failure comes from the residual homoscedasticity and the residual autocorrelation tests. It could be due to the fact that smaller petals are more difficult to measure. But another explanation, as it was mentioned above, could be that we are missing some relevant variable in the model.
We could solve the problem in two ways: 1) if the problem is the lack of an explanatory variable, we would need to look for it, include it in the model, and perform all the tests again. 2) If the problem comes from error associated with experimental techniques of sampling, or to the own mathematical features of the variable, the best thing to do would be transforming our variable. In the next session, mainly devoted to the General Linear Model, we will go into details about efficient ways to transform our variables.

Multiple linear regression

We are not going to get into much details about multiple regression (step selection of variables, etc.) since it is an issue that integrates well in the forthcoming section devoted to the General Linear Model. Just remember that the assumptions needed to be met by our data, and the ways to test them are the same as those detailed above except one additional. This additional one is the lack of co – linearity between predictor variables. In session 3 we will see how to deal with this.
In a multiple linear regression we simply have more than one predictor variable. The code to do it consist in adding variables with a "+" sign to the formula. Here is an example with the same data set:

```
> model2<-lm(Petal.Width~Petal.Length+Sepal.Length+Sepal.Width,data=iris)
> summary(model2)

Call:
lm(formula = Petal.Width ~ Petal.Length + Sepal.Length + Sepal.Width,
    data = iris)

Residuals:
     Min       1Q   Median       3Q      Max
-0.60959 -0.10134 -0.01089  0.09825  0.60685

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.24031    0.17837  -1.347     0.18
Petal.Length  0.52408    0.02449  21.399  < 2e-16 ***
Sepal.Length -0.20727    0.04751  -4.363 2.41e-05 ***
Sepal.Width   0.22283    0.04894   4.553 1.10e-05 ***
---
```

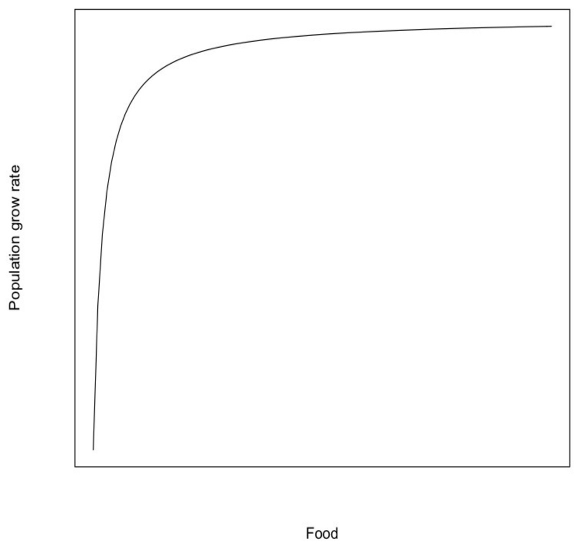
```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.192 on 146 degrees of freedom
Multiple R-squared: 0.9379,     Adjusted R-squared: 0.9366
F-statistic: 734.4 on 3 and 146 DF,  p-value: < 2.2e-16
```

1.3 Non – linear regression (file R.S1.4)

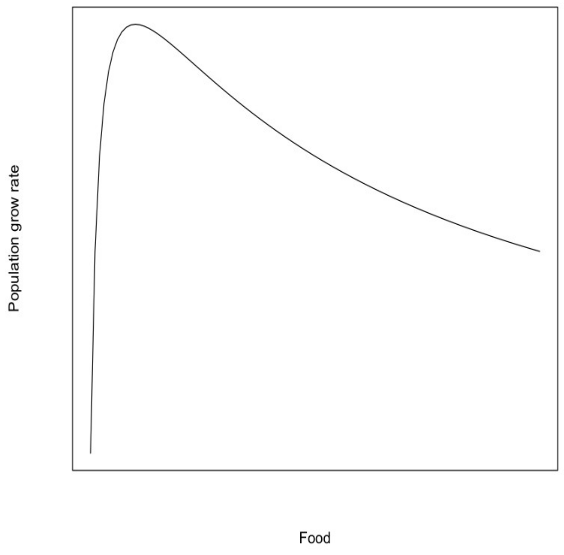
The assumptions underlying non – linear regression are very similar to those for linear regression, except that the function relating the predictors and the response variable is not linear. The techniques for fitting non – linear regressions are potentially the same as for linear regression. But, because we deal with more complicated functions and often with scarce data, things can get more complicated.
Usually, we use non – linear regression when we have a priori knowledge about the phenomenon that we are studying. We choose a non – linear function that we know that represents well the dynamics of such phenomenon.
We are going to see an example of the fit of two different non – linear functions to the same data set, so we will separately fit two models to the same data. This will give us the opportunity to introduce a basic technique of model selection. We are going to decide which of the two models fits the data better.

Fitting numerical responses of rotifers to Holling type II and type IV equations

Rotifers are planktonic invertebrates that feed on microorganisms, mainly, microalgae. In population ecology, the function that describes the population growth of an organism with respect to the amount of food ingested is termed “numerical response”. Holling developed a general mathematical model describing an archetypical numerical response, which is nowadays named “Holling equation” or “type II numerical response”. This model is defined by a monotonically increasing function that decreases its slope till reaching a saturating, stable point, when the organisms can not grow faster even if there was more food available. An hyperbola, in sum. This is how it looks:



In contrast, the type IV numerical response does not show saturation of the response to the increase in food. Instead, the population growth decreases. This could be due to different reasons, namely, a group defense strategy of the prey, or the presence of toxins in the prey. So this is how a type IV numerical response function looks like:



The function for the Holling type II equation has several possible versions, this form is what we are going to use, to determine the population growth rate:

$$\frac{\lambda_{max}(f-f_0)}{K_f+f}$$

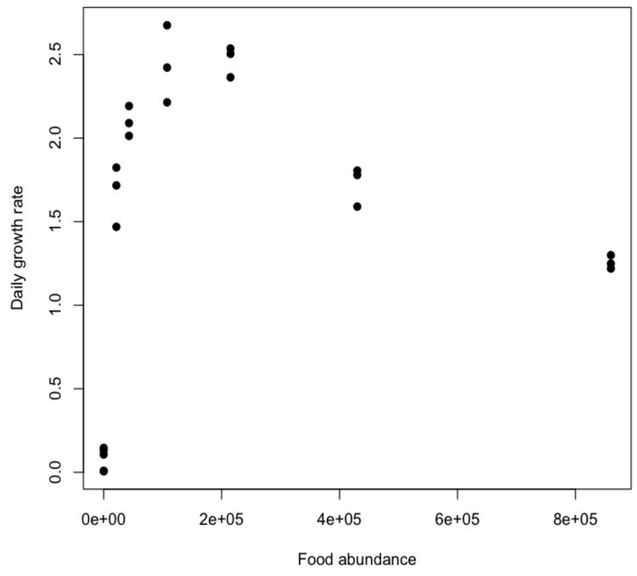
Where the parameters are: λ_{max} , the maximum growth rate, defining the upper asymptote. K_f , a half saturation constant. f_0 is the minimum amount of food necessary for positive growth, and *f* is the food, our independent variable.
And, for the type IV numerical response, we have the following equation defining the population growth rate:

$$\frac{\lambda_{max}(f-f_0)}{(K_f+f)(1+(i\times f))}$$

Where there is only one new parameter: *i*, which is defines the negative slope after the maximum of the function.
Our data consist in population growth rates of rotifers that were fed with constant amounts of food. So we are going to have a look at our data, fit these two non – linear functions, and choose the best one. First we load and plot the data. Food abundance are biovolume units.

```
> dat<-read.csv2("RotiferNR.csv",sep=";")
> head(dat)
   X FoodN          rN
1 24     0 0.106938130
2 25     0 0.145855274
3 26     0 0.005994843
```

```
4 27     0 0.009106229
5 28     0 0.131900895
6 29 21500 1.823847637
plot(dat$FoodN,dat$rN,xlab="Food abundance",ylab="Daily growth rate",pch=19)
```



The data look like they would better fit a type IV equation, but we better demonstrate it with statistic techniques.

First, we fit the two models. We are going to use the most basic fitting technique, which is the minimization of the squares sum, same as for the linear regression. But this is the version called non – linear minimum squared sum, or non – linear least squares. The function in R that does this task is nls(). The most important arguments of this function are the mathematical expression of the function, and an initial tentative value for all the parameters, provided in a list. We can guess the majority of these initial values by looking at our data. Otherwise, we could simulate our function varying certain parameters and see more or less which values give us a reasonable fit to our data. It is not a big problem if we are wrong with the initial value of one or two parameters. These is the code to fit the type II model:

```
> typ2<-nls(rN~(rmaxN*(FoodN-FoodminN))/
(KfN+FoodN),start=list(rmaxN=0.9,FoodminN=10000,KfN=20000),data=dat)
> summary(typ2)

Formula: rN ~ (rmaxN * (FoodN - FoodminN))/(KfN + FoodN)

Parameters:
          Estimate Std. Error t value Pr(>|t|)
rmaxN       1.9684     0.1375  14.314 5.69e-12 ***
FoodminN  -39.4740   173.9305  -0.227    0.823
KfN       973.7863  3579.3209   0.272    0.788
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4273 on 20 degrees of freedom
```

```
Number of iterations to convergence: 15
Achieved convergence tolerance: 8.619e-06
```

As we can see, among the three parameters, only the maximum growth rate is significant. When many parameters are not significant, this probably means that we chose the wrong mathematical function. Let's see the fit to the type IV model:

```
> typ4<-nls(rN~(rmaxIVN*(FoodN-FoodminIVN))/((FoodN+KfIVN)*(1+
(iN*FoodN))),start=list(rmaxIVN=1.5,iN=0.000003769,FoodminIVN=10000,KfIVN=5806),data=dat)
> summary(typ4)

Formula: rN ~ (rmaxIVN * (FoodN - FoodminIVN))/((FoodN + KfIVN) * (1 +
    (iN * FoodN)))

Parameters:
             Estimate Std. Error t value Pr(>|t|)
rmaxIVN     4.104e+00  4.563e-01   8.994 2.82e-08 ***
iN          2.546e-06  5.267e-07   4.835 0.000115 ***
FoodminIVN -6.636e+02  5.302e+02  -1.252 0.225888
KfIVN       3.195e+04  7.319e+03   4.366 0.000333 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1426 on 19 degrees of freedom

Number of iterations to convergence: 8
Achieved convergence tolerance: 2.191e-06
```

Here, most of the parameters are significant, except the minimum food. When a parameter is not significant, while many others are, this usually means that we do not need the non – significant parameter in our model. So we should simplify the model formula removing the useless parameter. Because this is just an example, we are not going to do it here.
The number of iterations to convergence is lower in the type IV model. This could mean that the technique was able to find better parameter estimates with less effort, but this also depends on the initial values provided for the parameters.
The residual standard error is lower in the type IV model. This might lead us to think that this model is better because there is less unexplained variance from the data. But, this, by itself is not what the statisticians consider that demonstrates that a model is better. Because the model of the type IV equation has one more parameter, this would by chance itself increase the amount of variance explained. So we need a method of model selection that considers the two aspects: first, it is better to explain more variance, but, second, it is not good to have many parameters, so increasing total number of parameters should be penalized. Parameters whose inclusion adds little contribution to the explained variance are not recommended. For this purpose was developed the Akaike Information Criterion (AIC), which takes into account the likelihood of the model, and the total number of parameters. This way of selecting models is based in the principle of maximum parsimony, that states that we do not need more complicated models (more parameters) to explain our data, if they do not add enough relevant information.
So, we need to know the AIC of each model, the lowest the AIC, the better the model. In this case, we will use a version of AIC named “corrected AIC”, which was developed for

small sample sizes (n = 15 – 60) like ours. The R function is the AICc(), from the *qpcR* package:

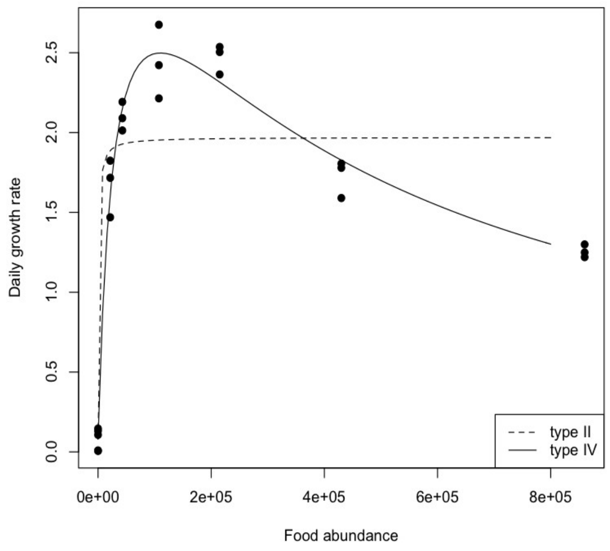
```
> require(qpcR)
> AICc(typ2)
[1] 32.21128
> AICc(typ4)
[1] -16.48615
```

The type IV model seems clearly better. But, we are not going to stop here. From a statistical point of view, it is better if, besides showing a clear different between the AIC values, we could show, somehow, whether if this difference is significant or not. For this purpose, we can apply a likelihood – ratio test to compare both AICc values (the AIC values are basically corrected likelihood units).
But there is one important point here. We can only use this test to compare "nested" models. What are nested models? We may say that they are "models of the same family". More correctly, models are nested when among them, one can be expressed as a particular case of the other. If you look at our formulas, it is clear that our models are nested, because the type II model could be considered as a particular case of the type IV model when the parameter *i* is equal to 0.
So here is the likelihood – ratio test, with function anova() (don't be confused here, this function is not performing analysis of variance, it has this name because it generates its output with a table of the same format as the anova tables):

```
> anova(typ2,typ4)
Analysis of Variance Table

Model 1: rN ~ (rmaxN * (FoodN - FoodminN))/(KfN + FoodN)
Model 2: rN ~ (rmaxIVN * (FoodN - FoodminIVN))/((FoodN + KfIVN) * (1 + (iN * FoodN)))
  Res.Df Res.Sum Sq Df Sum Sq F value    Pr(>F)
1     20     3.6523
2     19     0.3865  1 3.2658  160.54 1.029e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

As we can see, the difference in AIC is significant, so we definitely choose the type IV model.
In the following plot, we represent graphically the fitted curves and the data, so we can clearly see how the type IV model fits better the data:

```
> typeII<-curve((1.97*(x-(-39)))/(974+x),from=0,to=800000)
> typeIV<-curve((4.1*(x-(-664)))/((31950+x)*(1+(0.00000254*x))),from=0,to=800000)
> plot(dat$FoodN,dat$rN,xlab="Food abundance",ylab="Daily growth rate",pch=19)
> points(typeII,type="l",lty=2)
> points(typeIV,type="l")
> legend("bottomright",c("type II","type IV"),lty=c(2,1))
```

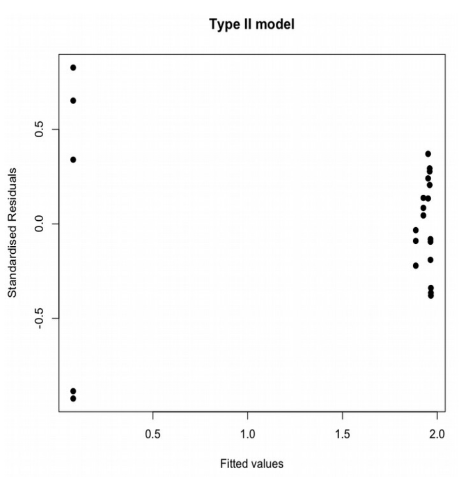


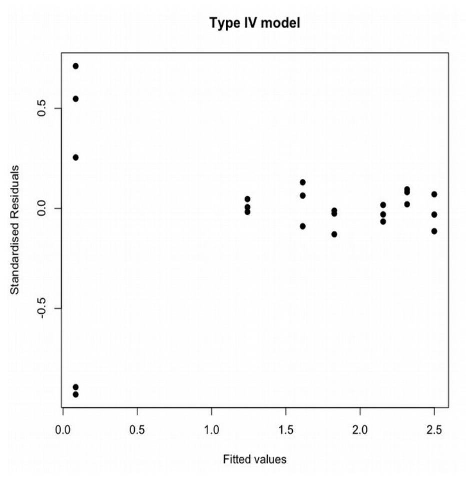
The type II model is not suitable for these data because it does not capture the decrease in growth rate after reaching the maximum, instead, it shows a saturating point. A few comments about the fitted parameters: it is clear that a negative value for the minimum food does not make biological sense. Additionally, t is not significant even in the type IV model. So we can eliminate this parameter for the model. The estimated values for the maximum growth rate are probably a bit overestimated in the type IV model, because this function projects the growth rate to higher values than the actual values that it shows in the data, as if there was a maximum value that the data can not reach because of the negative effect of food. However, In reality, we know that the maximum values shown in these data are close to the real maximum possible values.

How to select between models if they are not nested?

In this case we would keep with the model of lower AIC. But we can not estimate if the difference between AIC is significant. Also, having a look at a plot of fitted values versus residuals could help. Remember that residuals should follow a normal distribution, and in this kind of plot, they should to be distributed homogeneously at different distances around a hypothetical flat line. Here is how it looks like for our two models above:

```
> res2<-residuals(typ2)
> fit2<-predict(typ2)
> res4<-residuals(typ4)
> fit4<-predict(typ4)
> STres2<-res2/fit2
> STres4<-res4/fit4
> plot(fit2,STres2,pch=19,main="Type II model",xlab="Fitted values",ylab="Standardised
 Residuals")
> plot(fit4,STres4,pch=19,main="Type IV model",xlab="Fitted values",ylab="Standardised
 Residuals")
```





Both plots show a “flat” relationship. But residuals from type IV model are clearly more homogeneous. However, there is a problem common to both models, which is that there are larger residuals for the lower values of growth rate. This makes sense because lower growth rates show more variation. But this problem can not be solved for the residuals. This feature of the residuals suggests weighting the data, as we will explain below. This may improve the fitting of the models, but not solve the problem of the residuals.

What to do when fitting goes wrong with nls()

Here we are going to comment, without going into too much depth, a few features that can improve our fitting with this function. And if the fitting goes definitely wrong, what optional methods we have.
One feature of nls() that we did not use, but is suggested looking at the distribution of variance in the data and the plot of residuals, is adding weights to the data. What weights do is balancing the importance of each data during the fitting process. We did not mention it, but it is also possible to use them with the linear regression functions. The argument is very simple: *weights*, and its value is a vector with the weights. A very common way of weighting is by the inverse of the variance, or the standard deviation. Then, the weights will be lower in those data with higher variance. This gives less importance in the process to those data whee there is more uncertainty about their actual value. This is what we could have done in our examples.
One thing that can fail in nls() is about the minimum step size of the iterations. If this step is too large, the iterations might not find the best value for the estimation, because they are not precise enough. We can decrease the value of the step size with the argument *minFactor*, which is inside the function nls.control(), that can be used inside the function nls().
There is also the possibility that the maximum number of iterations is too low. Then, we can increase it as well, with the argument *maxiter*, located as well inside the nls.control() function. Here is an example of code showing how these two arguments are used (although in out type II model would not produce any change):

```
> typ2<-nls(rN~(rmaxN*(FoodN-FoodminN))/
(KfN+FoodN),start=list(rmaxN=0.9,FoodminN=10000,KfN=20000),data=dat,nls.control(maxiter=1
00,minFactor=1/5000))
```

If still, the nls() does not work, the reasons are that our data are good enough (few data or not matching the functions we are trying to fit) or because the model is too complex, it has too many parameters.
Before going to an alternative, stronger, fitting technique, we may try to fit each parameter alone with nls(), giving some tentative values to the other parameters. Then, we certain estimate for that parameter, add it to the model expression and go the next parameter. And so on until, after several rounds, the estimates of the parameters do not get better. This solves the problem of having too many parameters in the model.
If none of the things above work, then we have to go to more powerful methods of parameter estimation than the non – linear least squares from nls() function. One of them is the Nelder – Mead algorithm applied to maximum likelihood estimates. This can be done with the function mle(), from package *stats4*. The maximum likelihood gives us the quality of fitting of the model, in a similar way as the sum of squares. But the Nelder – Mead algorithm is a more powerful optimization method at searching the parameter values than the iterations performed by the least squares algorithm from nls().
Another methods, even less likely to fail than the Nelder – Mead algorithm are the “simulated annealing” algorithms. It is less accurate in the estimates, but their estimates could be further refined (if needed) by applying the mle() function. There are many functions that implement these algorithms in R, one of the most basic ones is the function anneal() from the package *likelihood*. However, the code is a bit more complicated for a beginner because we need to create a function or the model formula. Here is an example with the Gompertz growth model (Script *Example anneal*). First, we load the data, which are mollusk sizes with the corresponding time.

```
> ##load data
> dat<-read.csv2("postlarva tmn.csv",sep=",")
>
> ##Packages
> require(likelihood)
>
> ##Remove outliers: 154, 188, 246, 259, 341
> dat2<-dat[-(c(154,188,246,259,341)),]
```

Now, we create a user – defined function for the Gompertz model:

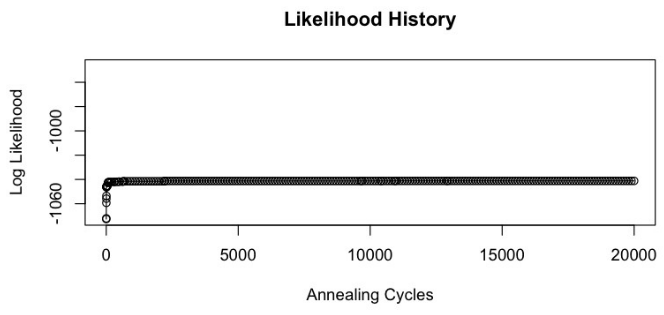
```
> #Function for Gompertz model
> modGom<-function(sizeInf,k,t0,tme1){
+       postlvsize<-sizeInf*exp(-exp(-k*(tme1-t0)))
+       return(postlvsize)
+       }
```

Now, the initial parameter estimates (the time they do not need to be accurate). The lower and upper bounds for parameter search, and the value for the argument “var”, that will be further used in the anneal() function.

```
> #Initial parameter estimates
> parIN<-list(sizeInf=30,k=0.02,t0=40)
>
> #Lower and upper bounds for parameater search
> parlow<-list(sizeInf=15,k=0.01,t0=20)
> parhigh<-list(sizeInf=50,k=0.5,t0=60)
>
> #Object for the further "var" argument
> var<-list(tme1=dat2$tme1,x=dat2$postlvsize,mean="predicted",sd=8.41,log=T)
```

Now, the simulated annealing itself, with function anneal(), and showing the optimized parameters and AIC value for the model at the end. You will get a plot with the increase in likelihood at each step of parameter iteration. Notice that the likelihood is to be maximized during this optimization process (not minimized, as it is the case for the minimun sum of squares optimization).

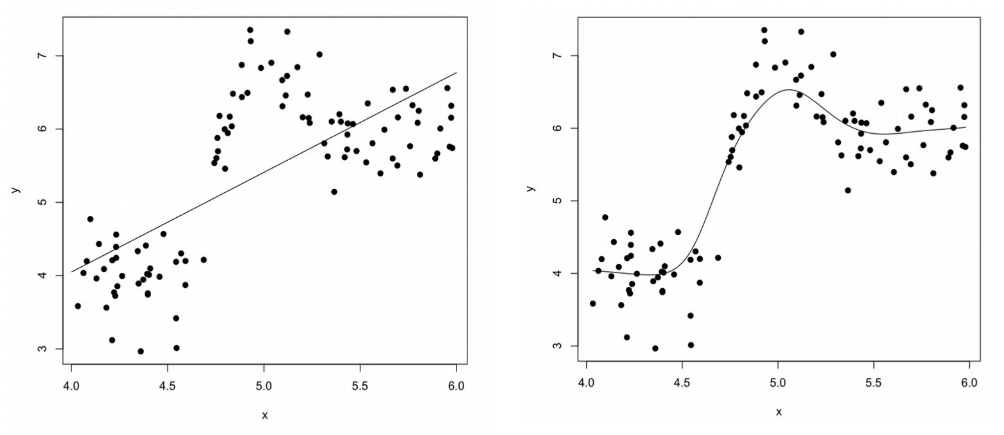
```
> #SIMULATED ANNEALING
> resultann<-
anneal(modGom,parIN,var,dat2,par_lo=parlow,par_hi=parhigh,dnorm,dep_var="postlvsize",max_
iter=20000)
>
> #Optimized parameters
> resultann$best_pars
$sizeInf
[1] 34.41218

$k
[1] 0.03516416

$t0
[1] 44.11132

>
> #AIC for the model with these parameters
> resultann$aic
[1] 2088.382
```



```
Completed cycle  20000.00  of  20000
Current temp:  1.8
Current best likelihood:  -1041.19
Goodness of fit - regression of observed on predicted:
Slope:    1   R2:   0.97
AIC corr:  2088.45
```

1.4 Non – parametric regression (file R.S1.5)

The methods for non – parametric regression have shown recent spread among several scientific disciplines. The major reason for this was the advance in computer power. These methods use very complex calculations that are performed very fast with our actual computers. They are very good tools for data exploration, and they are the basis of the fitting process of the generalized additive models, that we will further introduce in this course.
These methods, despite its name, do not consist in fitting functions “without parameters” to our data. The kind of functions that they fit are very flexible and complex functions, often with many parameters, that can adapt very well to the variations shown in our data. What “non – parametric” means here, could be more properly said “free of rigid parameter structure”. As we have seen in our previous examples, the linear functions or the type II and type IV models of numerical responses have a rigid “skeleton” that is determined by its simple formulation and the parameters.
This is an example with fake data comparing the results of fitting a parametric linear function (on the left) and non – parametric function (on the right, a polynomial function of degree 2) to the same data.



The non – parametric functions are, hence, more flexible, and can better adapt to the actual variations shown by the real data, because they are smoothed curves of the data. The main point is to keep the adaptation to this variation in a compromise between over smoothing and under smoothing. Under smoothing (the same as overfitting) means that the function would adapt to features that are specific of our sample data, and not a general feature of the phenomenon under study. Over smoothing means that the fitted function represents only very general features of the data. This compromise between under smoothing and over smoothing is also called the trade off between bias and variance, respectively.
Before getting into an example of non – parametric regression, there are a few assumptions and concepts that we should learn:

- Non – parametric regression follows the same assumptions than the linear and non – linear regression, except that the unknown function to fit has one only assumption to meet: being a smooth function.
- A key parameter to find is the bandwidth, usually represented as *h*. This parameter defines the width of an interval around each point of the data. Within this interval, the data

values will be smoothed and fitted to a curve. The non parametric curve will be constituted by all these curves fitted to the smoothed intervals.
You can clearly understand that, if *h* is high, the overall curve will be strongly smoothed, and the curve will represent only general features of the data. If *h* is small, the curvewill follow very close the small variations in the data points.
- There are several methods to perform the regressions inside the intervals defined by *h:* moving average (the simplest, not very common to be used), weighted moving average or Kernel, local polynomial regression, splines (weighted least squares) and others.
- Several of the methods above are weighted. This means that not every point inside the interval defined by *h* has the same importance. There are specific functions to weigh the values. Among the most used are the Gaussian, Epanechnikov, Triangular... They usually give more importance to the central point in the interval, and this importance decreases towards the extreme of the intervals.

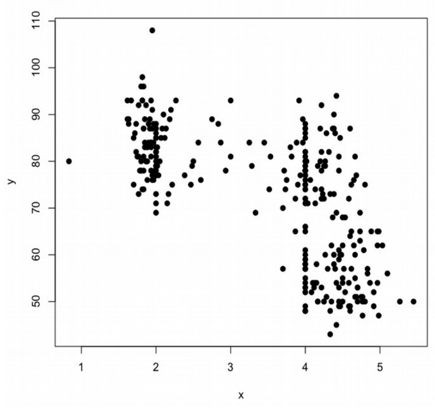
The value of *h* could be assigned arbitrarily, but it is better to use an objective method, because the parameter *h* is the key parameter of the model to determine the compromise between bias and variance. There were developed many methods for the objective estimation of *h* (they are called as well methods for estimating the complexity of the model):

- Based in the trade – off between MSE (measure of the local error in the interval) and MISE (measure of the global error of the data). These are called *Plug – in* methods, and exists a variety of them.
- Based in cross validation: the least squares cross validation and the generalized cross validation.
- AIC.
- Bootstrap methods
...

We are going to work now with a simple example of non – parametric fitting. For this purpose, we will use the data set available in R called *geyser*, from the package *MASS* (installed, but not loaded by default). This data set contains data on the duration of the eruptions of the geyser called "Old faithful", from Yellowstone Park (USA) during 15 days. First, we load the packages with the necessary functions, the data, and we plot the data to have a look at them:

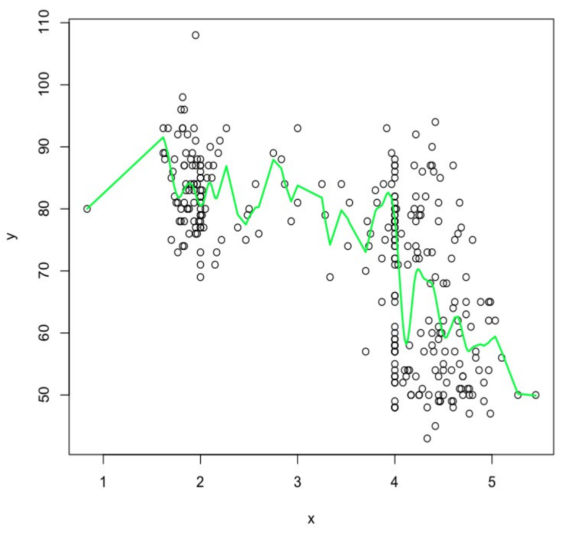
```
> require(MASS)
> require(KernSmooth)
> require(pspline)
> require(locpol)
> data(geyser)
> x<-geyser$duration
> y<-geyser$waiting
```

```
> plot(x,y,pch=19)
```



Now, we are going to fit a non – parametric curve with a spline method. The spline method uses polynomial functions of cubic degree, minimum.

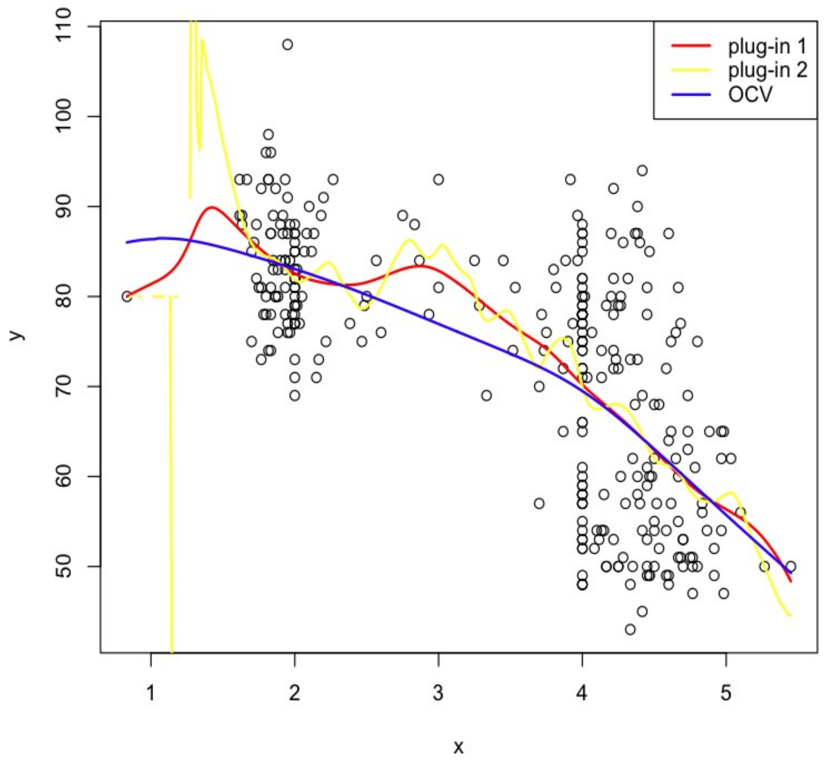
```
> spline<-sm.spline(x,y)
> plot(x,y)
> lines(spline,lty=1,lwd=2,col="green")
```



From my point of view, the cubic degree is already too much degree, the spline seems to be a bit under smoothing, or overfitting.
We are going to try now with a local polynomial method, with polynomia always of degree 2, and compare different methods of selection of *h*. Then, we will conclude which one is better for these data.

```
> #Plug - in method 1
> h1<-dpill(x,y)
> h1
[1] 0.2383502
> fit1<-locpoly(x,y,bandwidth=h1,degree=1)
> #Plug - in method 2
> h2<-pluginBw(x,y,deg=1,kernel=gaussK)
> h2
[1] 0.08805326
> fit2<-locpoly(x,y,bandwidth=h2,degree=1)
> #Ordinary cross validation method
> h3<-regCVBwSelC(x,y,deg=1,kernel=gaussK)
> h3
[1] 0.739138
> fit3<-locpoly(x,y,bandwidth=h3,degree=1)
```

The Plug – in method 2 is, in principle, the closest to under smoothing or overfitting, and the cross validation method, the one that most “oversmooths”. Let's see how they compare in a plot:



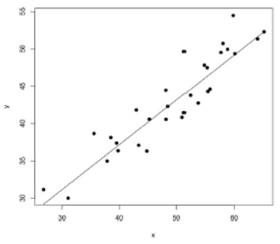
Our guesses were true regarding the values of *h*: the plug – in of the second type is clearly overfitting, and doing strange things in areas where there are no data. So, for these data, it seems that a local polynomia of degree 2 is the best choice. And, from my point of view, a bandwidth selector based in cross – validation is the best. Because it does not overfit and it does not depend so much on a single point in areas where there are few data (as the other two methods do with the first point, for instance).

Summary of regression

Linear regression

Fits a straight line of equation **y = ax + b**

y is the dependent, predicted or response variable
x is the independent or predictor variable
a and **b** are the parameters: **a** the slope, **b** the intercept



Assumptions

Linear relationship between **x** and **y**

Normal distribution of residuals

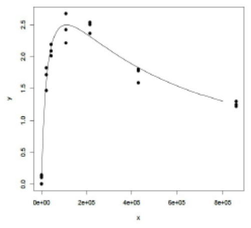
Homogeneity of residual variance (homoscedasticity)

Independence of residuals

Non linear regression

Fits a non linear equation like $y = \frac{r_{max} x}{K_s + x}$ $y = \frac{1}{1+e^{-x}}$ ...

y is the dependent, predicted or response variable
x is the independent or predictor variable
Parameters differ among equations



Assumptions

Non linear relationship between **x** and **y**

Normal distribution of residuals

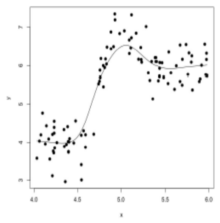
Homogeneity of residual variance (homoscedasticity)

Independence of residuals

Non parametric regression

Fits a smooth, complex equation with potentially many parameters, the equation is not important, but the fit to the data

y is the dependent, predicted or response variable
x is the independent or predictor variable



Assumptions

Smooth relationship between **x** and **y**

Normal distribution of residuals

Homogeneity of residual variance (homoscedasticity)

Independence of residuals

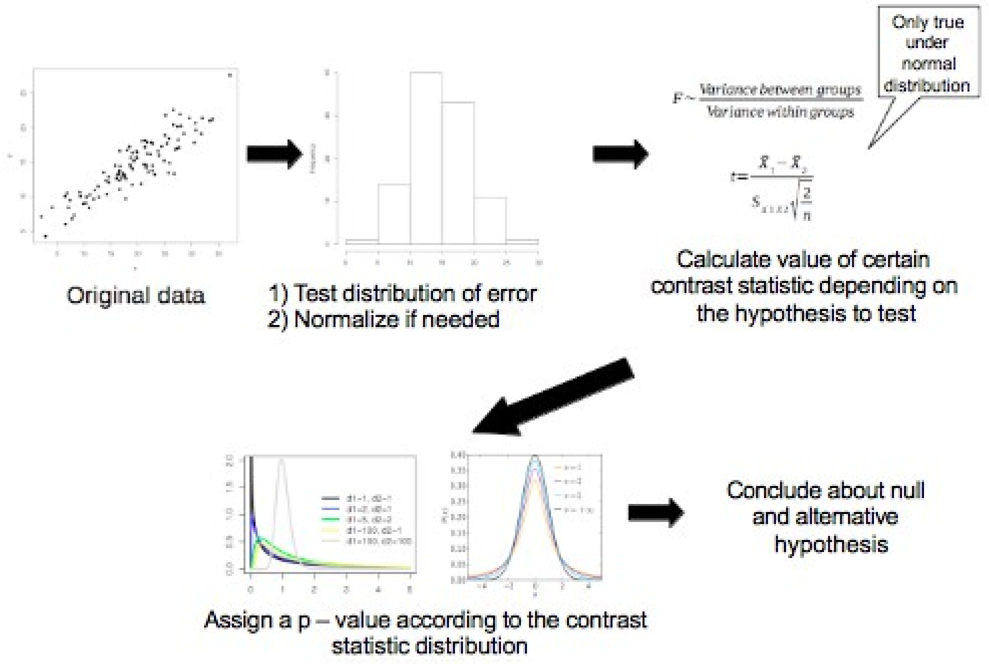
Section 2: Contrast of hypothesis

Contents

2.1 Main frameworks for hypothesis testing

We will talk briefly about four common frameworks for making inference from data, in order to learn the main differences between them. During this course, we will be using almost always parametric or frequentist statistics, but it is good to know the foundations of the other frameworks:

1) Parametric or frequentist statistics: they are the most commonly employed by researchers and taught in general statistics courses. They make the main assumption that the data to be analyzed follow certain distribution, most often the Normal distribution. Then, they calculate the parameters for that distribution. Based in the fact that those parameters belong to, often, a Normal distribution, a contrast statistic is calculated (Fischer's *F*, Student's *t*,...) which is a random variable whose distribution is known for the null hypothesis (H_0). The formula of the contrast statistic defines the hypotheses that is being tested (*F* is used for comparing variances, *t* for means, etc.)
Inference is made after assigning to the calculated value of that contrast statistic the probability of belonging to the known distribution of that contrast statistic under the null hypothesis (H_0). The limit value of this probability is, usually, by consensus $\alpha = 0.95$.
It is supposed that the null hypothesis reflects the “most objective” point of view. Then, the null hypothesis states that “everything you are comparing is the same” and the differences found between the observed and expected are due to chance. So, the variation is random. Parametric statistics test the probability of data given the null hypothesis is true.
The methods for inference were constructed in a conservative way, so that rejecting the null hypothesis is difficult.
One of the limitations of the scientific method applied with frequentist or parametric statistics is that the null hypothesis could be rejected, but it is never accepted. So, if the test yields a significant *p* – value, we learn that there is evidence to say that H_0 is false. But if the test yields a non – significant *p* – value, we are not allowed to say that H_0 is true. We simply say “There is no evidence to say that H_0 is false”. This comes from the hypothetic – deductive scientific method.
Here is a schematic representation of parametric or frequentist methods:


Only true under normal distribution
$F \sim \frac{Variance\ between\ groups}{Variance\ within\ groups}$
$t = \frac{\bar{X}_1 - \bar{X}_2}{S_{X1X2}\sqrt{\frac{2}{n}}}$
Original data
1) Test distribution of error
2) Normalize if needed
Calculate value of certain contrast statistic depending on the hypothesis to test
Conclude about null and alternative hypothesis
Assign a p – value according to the contrast statistic distribution

2) Bayesian statistics: within this framework it is also assumed that data follow a distribution, but not any general distribution. The distribution of the data is constructed during the analysis departing from prior knowledge about the distribution of our data. The general procedure of bayesian statistics can be summarized in the following steps:

- Setting a probability model with distributions for all the quantities of interest in our case: the dependent variable, but also the parameters (mean, variances,…) of the distributions are random variables with their probability distribution. This is known as the *prior* distribution. And it is constructed based in prior knowledge about the phenomenon under study. The differences with parametric statistics here are that the distribution assumed in that other framework is a general one, usually the normal, and the parameters have fixed values.
- Conditioned by our present observed data, reinterpreting and recalculating the *prior* to construct a *posterior* model of distributions that can better fit our data.
- Evaluate the *posterior* model: how it fits our data, if the conclusions are reasonable, and how sensitive are our observed data to the assumptions that our model needs. During this process, the model could be used to predict further observations. It could be used some sort of contrast statistic, defining the hypothesis to test, based in the predictions from our model, so that we can calculate the probabilities of the values of that contrast statistic. But unlike in the parametric framework, the distribution of the contrast statistic comes from the posterior model (so it is conditioned by our actual data) it is not a variable with a fixed distribution.

The inference is made from an opposite point of view than parametric statistics: it is tested the probability of certain hypothesis about the value of a parameter (not necessarily the null hypothesis) given the data, or conditioned by the data. This comes from the formula of probabilities of the Bayes theorem:

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

Where P(A|B) is the posterior probability, or probability of the prior P(A) given the observed data (B).
The conclusion is made after calculating this probability value of P(A|B) and there is no consensus value for anything. Obviously, probabilities greater than 0.5 give more chances for your hypothesis being true than false.
Bayesian statisticians believe that the null hypothesis of “no differences” is not necessarily the most objective. Rather, prior information should be used to specify a hypothesis.
They also criticize that parametric methods work with distributions that are not seen in the data (like the distribution of a contrast statistic, for instance).

3) Monte Carlo statistics: in this case, there are no assumptions about an underlying distribution of the data. The distributions of the data are calculated by randomizing the data with techniques like the *bootstrapping*. More or less like “shuffling” your data.
So, in summary, distributions are generated from the observed data. There might also be

no data, but we know the functional form and parameters (for instance, we know that our process follows certain distribution, and we know the parameters).
Probabilities are calculated from these simulated distributions. Then, the inference is made in a similar way than for the parametric statistics.
These methods are conceptually very easy to understand, but they could become complex at the operational level when complex models are involved. Mainly, at the step of generation of random data.
We use these methods when we can not make any assumptions about the underlying distribution of the data, either because we have limited amounts of data, or we do not know anything about the phenomenon under study. These two reasons would prevent us of applying parametric or Bayesian approaches to our analysis.
* the statistic frameworks are not that rigid. Monte – Carlo methods, for instance, are often involved in Bayesian methods and sometimes also with parametric and non – parametric methods.

Here is a very simple example of a Monte – Carlo method:

Imagine we have two samples that we want to know if they were drawn from different populations. So, if they come from the same underlying distribution but with different parameters (mean in this case) for each population. In this case our dependent variable is one morphological index (skull cephalic index = (skull length/skull width)*100) of fossil skulls from cavern bears found in two different caves. We want to know how likely it is that they belong to different populations. But we have very limited data (n = 8 per sample). So we better make no assumptions about normality. We do not have either any prior information about the distribution of this measure in cavern bears (they are extinct, and there are no data available). So we can not apply directly a Bayesian approach either.
These are our data:

```
> cave1<-c(54.8,53.5,56.2,52.1,59.2,59.6,55.1,54.7)
> cave2<-c(55.4,56.7,54.5,57.3,55.7,55.7,57.8,59.1)
```

What we wanted to test is if the mean index values are different between our 2 caves. So we have to construct a contrast statistic about the difference. The Student's t statistic is such one, but we can not use it here because it needs the assumption of normality for the data from both populations, and we can not assume it here. So we are simply going to use as contrast statistic the absolute value of the difference between the means, and we will call it “MD”:

```
> MD<-abs(mean(cave1)-mean(cave2))
> MD
[1] 0.875
```

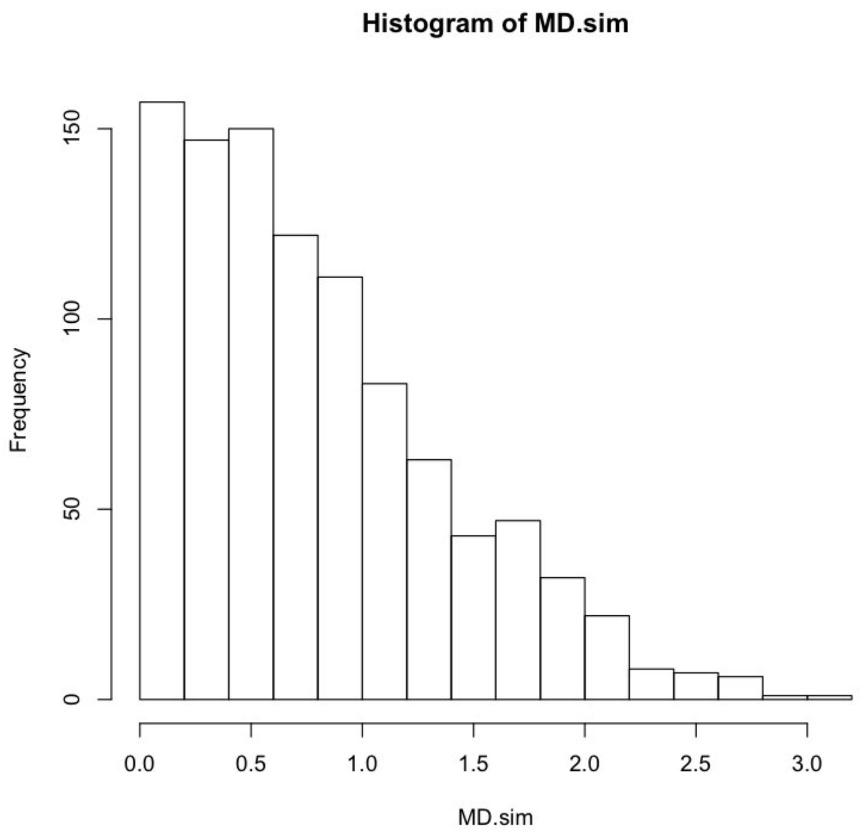
So this is our observed value of the contrast statistic. Now, we are going to simulate what would be its “null” distribution by “shuffling” our real values of skull index from both caves in order to create combinations of skull indexes with sample size = 8. Then, we will calculate their MD values from those combinations. Then, that distribution of MD values is going to be our “null” distribution, because samples from both caves are mixed in the same sample, so in principle, there should not be real differences between samples.
Then, we would calculate the probability of observing, within that null distribution, a value

greater than MD = 0.875, which is our observed value. So, we take the following steps: First, we use a loop to generate, for instance, 1000 simulations of MD values from “shuffled” combinations of our data:

```
> caves_1.2<-c(cave1,cave2)
> MD.sim<- c()
> for (i in 1:1000) {
+ cave1.s<-sample(caves_1.2,8,replace=TRUE)
+ cave2.s<-sample(caves_1.2,8,replace=TRUE)
+ MD.sim[i]<-abs(mean(cave1.s)-mean(cave2.s))
+ }
```

And this is how the simulations look like in a histogram:

```
> hist(MD.sim)
```



As we can see in this plot, the value of 0.875 seems highly probable. Let's calculate its probability:

```
> length(which(MD.sim>=MD))
[1] 391
```

There are 391 values out of 1000 simulations that are larger than 0.875. So the probability is 391/1000 = 0.391. So it is too high to reject the null hypothesis. We have to conclude then than skulls from both caves belong to the same population.

4) **Non parametric tests:** There are no assumptions about the distribution of your data and there is no construction of distribution for your data either. These methods do not use distributions, and so, no parameters (no mean, no assumptions about variances, etc.). Again remember not to mistake these techniques with the non parametric regression and related techniques: generalized additive models. In that context, "non parametric" means something like "free of rigid parametric functions" but parameters are still employed in those methods.
These methods use a contrast statistic with known distribution, like the parametric techniques.
But the contrast statistics do not need the assumption of normality, because in its calculation are not involved parameters from any distribution (means, variances, etc.). Instead, these are methods based, mainly, in ranking your data and comparing the ranks of the groups that you want to compare. For instance, the Mann – Whitney's *U* test has the following formula:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

Where n_1 and n_2 are the sample sizes of groups 1 and 2, and R_1 is the sum of al the ranks of the data from group 1. A value U_2 is also calculated equivalently with sample from group 2. The smallest value between U_1 and U_2 is chosen. Now, from this value, its *p* – value is calculated from a distribution of the *U* statistic. When sample size is large enough, the distribution of *U* can be approximated with a Normal distribution (then, the *U* value is normalized).
*Remember again that statistic frameworks are not that rigid. A mixture of them can be used when analyzing data. For instance, the Mann Whitney test can be applied as well to Normal data, in a similar way as *t* – test, but it is not as efficient.

Here is a brief example of calculation:

```
> #Data from group 1:
> y1<-c(6,7.5,4.25,5,3.2,7,5.3,6.1,3.6,4)
> #Data from group 2:
> y2<-c(8,7.1,6.7,4.5,5,3.8,7.7,6.9,8.1,5.2)
>
> #Join the data
> y12<-c(y1,y2)
>
> #Rank the data (just to see how they rank)
> rank(y12)
 [1] 11.0 17.0  5.0  7.5  1.0 15.0 10.0 12.0  2.0  4.0 19.0 16.0 13.0  6.0  7.5  3.0
[17] 18.0 14.0 20.0  9.0
```

Note that the value 5 is present in both groups, so there is a tie. This is solved by assigning the rank 7.5 to both cases (the average between rank 7 and 8).

```
> #Ranks for y1 and y2
> r1<-c(11,17,5,7.5,1,15,10,12,2,4)
> r2<-c(19,16,13,6,7.5,3,18,14,20,9)
```

We can see that values from group 2 seem to have greater ranks. Now, if we apply the formula for both groups:

```
> #Apply Mann - Whitney's U formula
> U1<-10*10+(10*(10+1)/2)-sum(r1)
> U1
[1] 70.5
> U2<-10*10+(10*(10+1)/2)-sum(r2)
> U2
[1] 29.5
```

The smallest *U* value is for group 2, so we use U_2 for calculating the *p* – value. Checking a table of limit values of *U* for α = 0.95 with samples sizes n_1 = 10 and n_2 = 10 is 23. Our values is greater than 23, so the groups have significantly different values.

Another very common non – parametric test consists in using, to calculate the contrast statistic, a ratio of observed and expected frequencies of a categorical variable. The distribution of this contrast statistic is often approximated with a Chi – squared distribution.

$$X^2=\sum\frac{(O_i-E_i)^2}{E_i}$$

Where O_i are the observed values of certain categories and E_i are the expect values (based in the calculated probabilities). These expected probabilities could be calculated assuming certain probability distribution (like in a typical goodness of fit test) or directly from our observed data, assuming a discrete uniform distribution. This is the typical case of “contingency tables” (in the non – parametric case).

	Male	Female	Total
Smoker	23	31	54
Non Smoker	39	34	73
Total	62	65	**127**

The row and column totals are called “marginal totals” and the total number of individuals is called “grand total”. Here, the expected probabilities of a uniform distribution would be calculated simply dividing the grand total by the number of cells. In this case, p = 0.25 for each cell.

The non – parametric framework is usually the less preferred. It is used when we have strong limitations in our data, usually due to small sample sizes.

2.2 Parametric and non parametric tests for hypothesis testing
(file R.S2.1)

The tests for hypothesis testing, under the frameworks of parametric and non parametric statistic, have three main steps, as we know from our previous section:

1) Assume (parametric) or not (non parametric) certain distribution for our data.

2) Based in the values of certain parameters of the distribution (parametric) or based in other quantities (ranks, observed-expected) calculate the value of a contrast statistic. The contrast statistic must be suitable for the hypothesis that we are testing. The hypotheses is implicit in the calculation of its value (formula).

3) Assign a probability to the contrast statistic based in its known distribution.

So, the only point that differs between the tests is the hypothesis being tested (i.e. the contrast statistic). In order to select a suitable contrast statistic, it is very important the difference between categorical or qualitative dependent variables and numerical dependent variables.
Now we are going to see a brief summary of common statistical test classified according to the criteria of a) the hypothesis to contrast, b) parametric – non parametric and c) quantitative – qualitative dependent variables. There are more tests than what is shown in this summary, but the essential contrasts are shown here. Most of other tests are alternatives or special cases of those shown in this summary.

A. Goodness of fit tests
A.1 Quantitative variables
A.1.1 Parametric

- Shapiro – Wilks: assumes the normality of the data and test the null hypothesis that data come from a Normal distribution. We have already used it and we will keep using it for a while.

A.1.2 Non parametric

- Kolmogorov – Smirnov: The one – sample version of his test compares one sample with a reference (continuous) distribution. We have seen an example with the fit to a gamma distribution in session 2.
- Pearson's chi – squared test for the goodness of fit: it uses a contrast statistic that compares the observed frequencies of a variable and its expected frequencies under the assumption of certain theoretical distribution. It can be used with contingency tables (frequency data distributed in categories). Example:

```
> dat<-seq(0,1,by=0.05)
> Obsv.p<-
c(0.38,0.4,0.43,0.45,0.45,0.48,0.5,0.53,0.52,0.51,0.54,0.59,0.65,0.7,0.8,0.76,0.81,0.84,0
.9,0.95,0.94)
> Theor.p.Norm<-sapply(dat,pnorm)
> chisq.test(Obsv.p,p=Theor.p.Norm,rescale.p=T)
```

```
	Chi-squared test for given probabilities

data:  Obsv.p
X-squared = 0.2866, df = 20, p-value = 1
```

B. Homogeneity
B.1 Quantitative variables
B.1.1 Parametric

- Stundent's *t* test: this contrast statistic is often used when comparing the homogeneity of means of two populations. The null assumption is that both means are equal. It can be used with related (dependent) samples or independent samples. It is used as well when we want to compare the mean of our data against a fixed value. On each case, the formula for the contrast statistic is slightly different. In this later case is often applied to test the significance of the slope of a linear regression, being the null hypothesis that the slope is equal to 0 (that would be, obviously, a non significant effect of our predictor on our response variable).
These are some examples of use of the t – test:

In independent samples:

```
> data(iris)
> pop1<-iris[iris$Species=="setosa","Sepal.Length"]
> pop2<-iris[iris$Species=="virginica","Sepal.Length"]
>
> t.test(pop1,pop2)

	Welch Two Sample t-test

data:  pop1 and pop2
t = -15.3862, df = 76.516, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.78676 -1.37724
sample estimates:
mean of x mean of y
    5.006     6.588
```

The difference between means is highly significant than 0 (the null hypothesis is rejected). In this case of independent samples we have to keep the default value of the “paired” argument (FALSE) which value would be TRUE only in the case of a test for related (dependent) samples. For instance, imagine that we have a measurement of % hematocrit of 15 athletes before and after a specific training. We want to know if the training has increased the level of hematocrit, but our samples are not independent (they belong to the same athletes):

```
> s1<-c(48.8,37.3,46.4,43.4,37.6,40.4,40.7,42.5,47.5,36.6,40.9,35.5,51.7,57.5,43.9)
> s2<-c(56.5,46.6,53.4,47.5,51.5,51.2,61.7,53.8,52.1,46.7,50.6,54.4,50.4,50.7,52.5)
>
> t.test(s1,s2,paired=TRUE)
```

```
      Paired t-test

data:  s1 and s2
t = -4.7797, df = 14, p-value = 0.0002936
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -12.449434  -4.737233
sample estimates:
mean of the differences
              -8.593333
```

Here, the null hypothesis is rejected as well, since the *p* – value is smaller than 0.05.
If we want to perform a t-test for the hypothesis of the mean being different than a fixed value, we have to specify this value in the argument ”mu”:

```
> mean(s1)
[1] 43.38
> t.test(s1,mu=50)

      One Sample t-test

data:  s1
t = -4.1838, df = 14, p-value = 0.000919
alternative hypothesis: true mean is not equal to 50
95 percent confidence interval:
 39.98633 46.77367
sample estimates:
mean of x
    43.38

> t.test(s1,mu=43)

      One Sample t-test

data:  s1
t = 0.2402, df = 14, p-value = 0.8137
alternative hypothesis: true mean is not equal to 43
95 percent confidence interval:
 39.98633 46.77367
sample estimates:
mean of x
43.38
```

So, for a real value of 43.38 for the mean, comparing it against a value of 50, the difference is found to be significant, and for a value of 43 is found to be non significant, as it could be expected.

- Fischer – Snedecor *F*: This test can be used for different purposes: homogeneity of means, homogeneity of variances, fit of a regression model. On each case, the *F* contrast statistic is calculated in a different way (ratio of variances, ratio of RSS, ...) but always follows the *F* distribution. There is a specific function in R for the homogeneity of variances, var.test(). The F test for the fit of a regression or linear model is performed by default in R when fitting a linear model.

- Levene's test: an alternative to F test for the variance homogeneity. In R, there is the function leveneTest() in the package *car*.
- Bartlett's test: another test for the homogenity of variances. In R, we have the function bartlett.test().

B. Homogeneity
B.1 Quantitative variables
B.1.2 Non parametric

- Kolmogorov – Smirnov: test for the homogeneity of 2 independent samples. Example:

```
> a<-c(10.01,11.18,8.09,11.45,12.69,14.64,6.28)
> b<-c(6.43,16.75,18.41,9.84,7.36,18.98,13.89)
> ks.test(a,b)

	Two-sample Kolmogorov-Smirnov test

data:  a and b
D = 0.4286, p-value = 0.5752
alternative hypothesis: two-sided
```

- Mann – Whitney: test for the homogeneity of 2 independent samples. In R there is a single function for performing this test and the Wilcoxon test (see below) for 2 dependent samples. They are not the same but they are equivalent. Example:

```
> wilcox.test(a,b)

	Wilcoxon rank sum test

data:  a and b
W = 18, p-value = 0.4557
alternative hypothesis: true location shift is not equal to 0
```

- Wilcoxon: test for 2 dependent samples.

```
> wilcox.test(a,b,paired=T)

	Wilcoxon signed rank test

data:  a and b
V = 7, p-value = 0.2969
alternative hypothesis: true location shift is not equal to 0
```

- Kruskal – Wallis: test for k independent samples. Example:

```
> c<-c(17.71,19.95,18.52,18.44,19.26,15.89,21.51)
> groups<-list(a,b,c)
> kruskal.test(groups)

	Kruskal-Wallis rank sum test

data:  groups
```

```
Kruskal-Wallis chi-squared = 10.5603, df = 2, p-value = 0.005092
```

B. Homogeneity
B.2 Qualitative variables
B.2.1 Non parametric

- Chi – square test for independent polytomous variables: it is basically the same as the one used with quantitative variables, because uses frequencies from a contingency table. The difference is that here the frequencies are of categorical variables. Example: with the data set Titanic, we are going to test whether the survival of Titanic adult passengers is influenced by the categories of two qualitative variables: sex and class. The data set comes in an array with four strata: for not surviving child, not surviving adults, surviving child and surviving adults.

```
> data(Titanic)
> Titanic
, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No

      Sex
Class  Male Female
  1st   118      4
  2nd   154     13
  3rd   387     89
  Crew  670      3

, , Age = Child, Survived = Yes

      Sex
Class  Male Female
  1st     5      1
  2nd    11     13
  3rd    13     14
  Crew    0      0

, , Age = Adult, Survived = Yes

      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
```

We have to use only the last matrix for our test:

```
> ct<-Titanic[,,Age="Adult",Survived="Yes"]
> chisq.test(ct)

	Pearson's Chi-squared test

data:  ct
X-squared = 220.3731, df = 3, p-value < 2.2e-16
```

So, the combination of factors influenced the probability of survival in adults.

- Fischer test for independent dichotomous variables: With dichotomous variables (i.e.: 2x2 contingency tables). It is better not to use the previous chi – squared test. We should better use the present one, the “Fischer exact test”. Example: with the same precious data, we would like to see if there were differences in survival between the 1st class passengers and the crew:

```
> ct2<-ct[c(1,4),]
> ct2
      Sex
Class  Male Female
  1st    57    140
  Crew  192     20
> chisq.test(ct2)

	Pearson's Chi-squared test with Yates' continuity correction

data:  ct2
X-squared = 160.2842, df = 1, p-value < 2.2e-16
```

So differences were found also in this case.

- Friedman's Anova for dependent polytomous variables: it is called “Anova” because of its use, similar as the formerly called “repeated measures Anova” (nowadays, a repeated measures Anova is a general linear model with individual as random factor, as we will further see). It could be used as well with quantitative dependent variables, as a sort of non – parametric repeated measures Anova. We will see an example of this later case: we have 20 wine tasters and 3 wines. We want to know if each wine is being consistently being rated with higher or lower scores than the other ones (we could also do it in the opposite sense: test if the wine tasters are giving consistently different rates to all the wines):

```
> data<-read.csv2("wine.csv")
> head(data)
  X Taste   Wine Taster
1 1  5.40 Wine A      1
2 2  5.50 Wine B      1
3 3  5.55 Wine C      1
4 4  5.85 Wine A      2
5 5  5.70 Wine B      2
6 6  5.75 Wine C      2
> friedman.test(data$Taste,data$Wine,data$Taster)
```

```
      Friedman rank sum test

data:  data$Taste, data$Wine and data$Taster
Friedman chi-squared = 8.4416, df = 2, p-value = 0.01469
```

- Cochran's Q test for dependent dichotomous variables: Version of the previous one for dependent variables with two possible outcomes. Example (directly from the function's help):
The dependent or response variable has only 2 possible values (1 or 0)

```
> require(RVAideMemoire)
> response <- c(0,1,1,0,0,1,0,1,1,1,1,1,0,0,1,1,0,1,0,1,1,0,0,1,0,1,1,0,0,1)
> fact <- gl(3,1,30,labels=LETTERS[1:3])
> block <- gl(10,3,labels=letters[1:10])
> cochran.qtest(response~fact|block)

      Cochran's Q test

data:  response by fact, block = block
Q = 10.8889, df = 2, p-value = 0.00432
alternative hypothesis: true difference in probabilities is not equal to 0
sample estimates:
proba in group A proba in group B proba in group C
             0.2              0.5              1.0

        Pairwise comparisons by using Wilcoxon sign test

        A       B
B 0.37500       -
C 0.02344 0.09375

P value adjustment method: fdr
```

Section 3: GLM

Contents

3.1 General Linear model I

The general linear model (GLM) is a frame for the statistical analysis and inference for the study of linear relationships between a response variable and one or several predictor variables. It is a generalization of simple and multiple linear regression in order to incorporate categorical factors and complex designs, like nested designs, random factors, etc. The response variable is often called the dependent variable and the predictor variables independent variables.
We depart from a sample of individuals that have one measurable property that is the dependent or response variable. We want to predict the values of that property from a linear combination of other variables, the independent variables.
The independent variables could be factors or covariates. Factors are variables structured in levels, for instance, three different types of diet. Covariates are not structured in levels and they are numeric, for instance, time is a common covariate in many general linear models. Their effect is always additive (this means that they can not interact with other independent variables, we will see later what does this mean).
However, this terminology is confusing and in some statisticians use the term "covariate" for any kind of independent variable.

Formulation

The general formulation for a linear model of this kind would be the following:

$$Y = ß_0 + ß_1 X_1 + \ldots + ß_n X_n + \epsilon$$

Where Y is a vector with the values of the dependent variable. $ß_0$ is the intercept of the model, $ß_1$... $ß_n$ are the vectors of the model coefficients for each independent or predictor variable (if the predictor is a factor, there will be $n - 1$ coefficients for that factor, being n the number of factor levels, if the predictor is a covariate, there is a single coefficient for it) X_1 … X_n are the vectors with the values of the independent or predictor variables (if it is a factor, each value is a level, and if it is a covariate, a single numerical value) and ϵ is the vector of the residual errors of the model.
The residual error is, as we already know, the difference between the observed values and the predicted or fitted values

$$\epsilon = Y - \hat{Y}$$

Assumptions

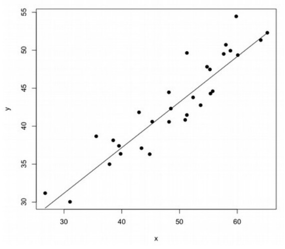
The assumptions are related with those of the linear regression, and are tested in the same way, except the homoscedasticity.

- Linear relationship (or linearizable).
- Independence of predictor variables. This can be tested with simple regressions between them or with the VIF (variance inflation factor) that we will discuss in the next example.
- Independence or no autocorrelation of the values of the response variable.
- Normal distribution of residual error.
- Homogeneity of variances. The homogeneity has to be between groups of factor levels, it

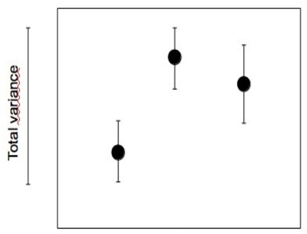
is related with the normality of the errors. It is usually tested with Levene's test.
- Homogeneity of slopes: only when a covariate is included. We will explain this in detail later.

There are two main steps when we analyze data with a general linear model:

A) Fit of the linear model: this is analogous to a linear regression fit. The values of the $ß_i$ coefficient from the equation above are obtained, and a *t* – test will tell us if these coefficients are significantly different of 0. Usually, if one or several model terms do not show any significant coefficient, we could remove them from the model. Then, our final model would have only significant terms. The fact that the model terms are significant simply mean that there is a relationship between the response variable and the predictors. Nothing else. Graphically, what we are doing is this:



B) Analysis of variance. The main purpose of a linear model in many cases is to detect significant effects of the factors, among factor levels. The linear model that we fitted in the previous step does not tell us anything about it (although, if the coefficients of a factor are highly significant, usually happens that the factor levels have significantly different values of the response variable). The analysis of variance will tell us if the differences between groups are significant. But we previously need to fit a linear model in order to accomplish with the assumptions under which the analysis of variance works. The differences in variance are analyzed with the Fischer – Snedecor *F* contrast statistic (remember, this is a ratio of total or between groups variance versus within groups variance). The effects of each predictor variable independently, are called **Main effects**. Graphically, what we are doing at this step is checking if the variances between factor levels like the following, are different:



After the Anova, we might be interested in comparing which levels inside a factor are significantly different from others in pair – wise combinations. This is called “post – hoc test”. Usually, they are performed with contrast statistic that test the difference between means (similar to a Student's *t*). The effects of these pairwise comparisons of factor levels are sometimes called **simple effects**.

"Ugly" example **(file R.S3.1)**

We are going to start with a simple example of linear model with the *iris* data set. If we look at the plot from the example of linear regression, we might think that there is a linear relationship that could predict the petal length from the petal width, with certain influence of the species. So, we are going to construct a linear model that aims to predict the petal length (our response or dependent variable) with petal width (a predictor or independent variable that is a numeric covariate) and the species (a predictor or independent variable that is a factor with three levels: *setosa*, *versicolor* and *virginica*). So the first things we do are loading the data and defining the model with the lm() function:

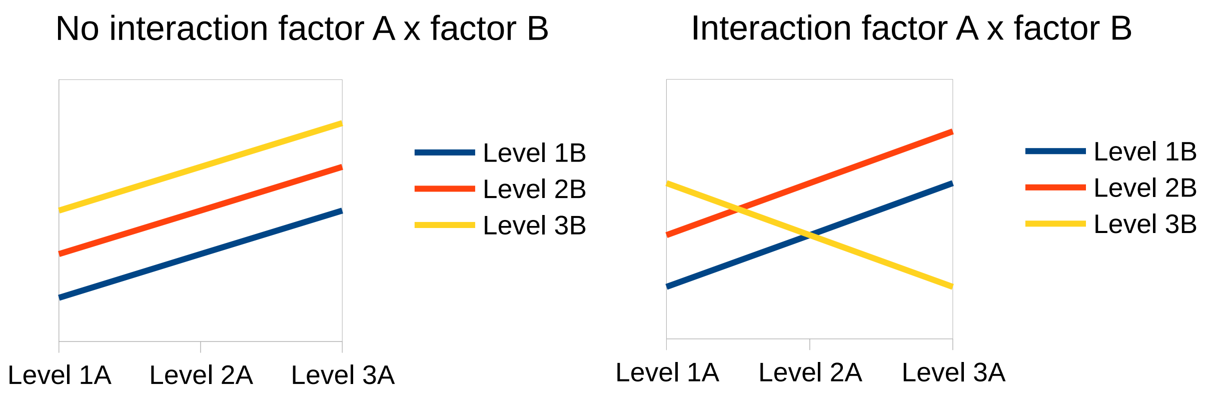
```
> data(iris)
> mod1<-lm(Petal.Length~Petal.Width*Species,data=iris)
```

It is time to talk about the interactions, an important term of the linear models that governs the way the predictors interact between them and with the response variable.
In the linear models, the type of interaction expected between variables is linear. "linear" in this context is equivalent to "additive". This means that if we increase n units in the value of the predictor, the value of the response will increase ßn units (remember ß is the coefficient of that predictor, like the slope of a linear regression). So, if we increase 2 units in the predictor, the response will increase $ß_0 + ß_1 2$ units, if we increase 3 units in the predictor, this yields an increase of $ß_0 + ß_1 3$ in the response... and so on. So this is a linear, additive relationship.
When two predictors that separately have a linear relationship with the dependent variable, if put together in an experimental design do not show a linear relationship with the response variable, we say that there is a **significant interaction** between the predictors**.**
Without interaction, the value of the response variable can be estimated through the additive effect of each predictor separately. Imagine that there is one increase of 2 units in the first predictor and 3 units in the second predictor. Then, the value of the response must be equal to $ß_0 + ß_1 2 + ß_2 3$. As you can see, just the addition of the effect of the two predictors (additive effect). So when the effect that we observe is not explained by an addition, rather it is a multiplicative effect, or other non – linear kind of effect, we say there is an interaction. It is the same as a synergy.
Here is an imaginary example of interaction: we study the effect of diet and temperature in the size at maturity of fish. We know that there is a linear positive effect of temperature in the growth of fish. Then, we have different diets, one control and others with different concentrations of a vitamin supplement. We know that these vitamin supplements have, as well, a positive linear effect on the size of fish. But now, imagine that these vitamin supplements are not equally effective across all the range of temperatures that we have. The higher the temperature, the faster the vitamins degrade, and then they will be less available for the fish. In this case, there will be an interaction between temperature and diet, in a way that with an increase in temperature, the positive effect of vitamins is reduced. Then, we can not predict the final effect of both variables as an additive effect, because the high temperature data do not fit the additive effect. In summary, the relationship between fish size and diet would not be linear if we include data from high temperatures.
The problem of having interactions is that they compromise our interpretation of the results from a linear model. With an interaction in the model, we can not interpret the effects of each predictor separately, because they are influenced by other predictors. So we can not

interpret our Main effects. What we can do is to check the Simple effects with a post-hoc test, or by looking at the model coefficients, and then try to see how the interactions are working at each level of the predictors. At some of these levels, there might be still a additive effect, and not so in others.
So, when we start fitting a model, we have to include, at the beginning, all the possible interactions. Later, we will test with the Anova if those interactions are significant or not. If they are not significant, we must remove them from the model formula (as we would do with any other model term).
An interaction is a term of the model formula that has its own coefficient as well, in fact, it is considered as a new predictor.

*A brief note on the homogeneity of slopes (if there were covariates in the model). Remember that a "pure" covariate, apart from being numeric, it is always additive, never interacts with other predictors. So if we want to include a covariate, we have to test its interaction as well. If it interacts, this means that the slopes of the additive effect of the covariate are not homogeneous between groups. There are some specific tests for this, but, testing directly the interaction term is valid, and the most common way to do it.

Here is a graphic explanation of what the concept of **interaction**:



These plots show how would look like the lines that go across the mean values of the dependent or response variable on each level of the factors. On the left, we have "Main effects" for both factor A and factor B. this means that from level 1 to 3 of factor A there is a positive effect on the response variable (all the three lines show this effect). There is also a positive effect of factor B, because each level of B shows a line that is "higher" than the other. However, these lines are all parallel. This means that the effects are all additive, and there is no interaction. On the right, we can see how the line of the level 3 of factor B is not parallel to the others. This means that there is an interaction of factor A x factor B (in this example yellow line is crossing the others, but it does not have to cross them to be an interaction, just not being parallel).

How do we code interactions in R? It is very simple. The symbol "*" indicates that we consider all the possible interactions between the variables that have that sign. The symbol ":" indicates that we will consider only interactions of order 2 (all the pairwise combinations of predictors). The symbol "^n" indicates that we only want to consider the

interactions from degree 2 to degree “n”. The symbol “+” indicates only additive effects, so no interactions at all. Look at the examples on page 3 to see how they are coded.
In our example, we have only two predictors, so only the interaction of degree 2 is possible, then it is not relevant which sign to use.
Now, let's have a look at the results of the model fit:

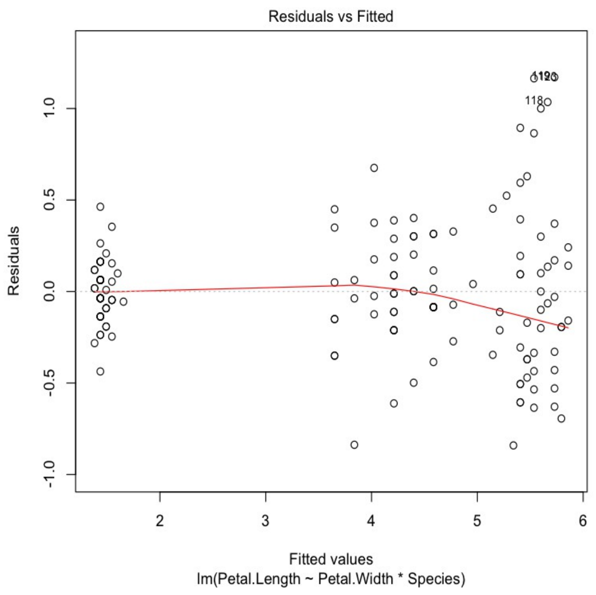
```
> summary(mod1)

Call:
lm(formula = Petal.Length ~ Petal.Width * Species, data = iris)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.84099 -0.19343 -0.03686  0.16314  1.17065 

Coefficients:
                              Estimate Std. Error t value Pr(>|t|)    
(Intercept)                     1.3276     0.1309  10.139  < 2e-16 ***
Petal.Width                     0.5465     0.4900   1.115   0.2666    
Speciesversicolor               0.4537     0.3737   1.214   0.2267    
Speciesvirginica                2.9131     0.4060   7.175 3.53e-11 ***
Petal.Width:Speciesversicolor   1.3228     0.5552   2.382   0.0185 *  
Petal.Width:Speciesvirginica    0.1008     0.5248   0.192   0.8480    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3615 on 144 degrees of freedom
Multiple R-squared:  0.9595,	Adjusted R-squared:  0.9581 
F-statistic: 681.9 on 5 and 144 DF,  p-value: < 2.2e-16
```

In the main table are shown the values and statistical significances of the model coefficients. The intercept is significant. The coefficient for petal width is not significant. There are two coefficients for the predictor “species”. One is significant (for the species *virginica*) the other, for the species *versicolor*, is not. For the interaction, only the coefficient for *versicolor* is significant. As you may have noticed, the coefficients of a single predictor are always 1 if it is a numeric predictor (covariate) or n-1, being n the number of levels if the predictor is a factor structured in levels. Our factor “species” has three levels, so it shows two coefficients. This “n-1” number of coefficients is due to one restriction that the linear models have in order to make easier the fit of the model terms. They need to assume that one of the coefficients has value 1 or 0. Then, the others are expressed relative to that one. This level for which the coefficient is equal to 0 or 1 is called the “dummy variable”. The dummy variable is assigned in lm() by default. The value of its coefficient is, also by default, 0. these default settings can be changed, but we are not going to deal with this now.
So now we can better interpret our coefficients (forget for now whether I they are significant or not):
For petal width: an increase in one unit of Petal width increases 0.5465 units of petal length.
For the factor “species”: belonging to the species *virginica* makes your petal length increase 2.9131 times units more than belonging to *setosa* (the dummy variable), and belonging to the species *versicolor* makes your petal length to increase 0.4537 units more

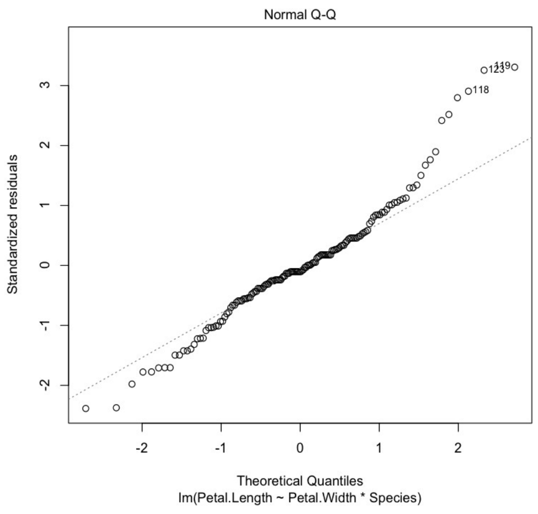
than belonging to the species *setosa*.
For the interaction: the effect in one unit of petal width if belonging to the species versicolor increases 1.3228 times the multiplication of their coefficients as individual effects (0.5465 x 0.4537). And the equivalent for the other interaction coefficient.
We already know how to interpret the remaining information from this output (remember the linear regression example).
Remember what we said above about the interactions and covariates. If the covariate has a significant interaction with the other predictor, we can not drive strong conclusions about its effect. Later, we will se with an Anova if the interaction is significant or not.
Now, we have to continue working with this model.
Before inferring if the effects of the predictors are significant, we have to test first, if the model agrees with the assumptions needed for linear models (normality and homoscedasticity). Before performing specific tests, we could have a look at relevant plots in order to see what the model residuals tell us.

```
> plot(mod1)
```

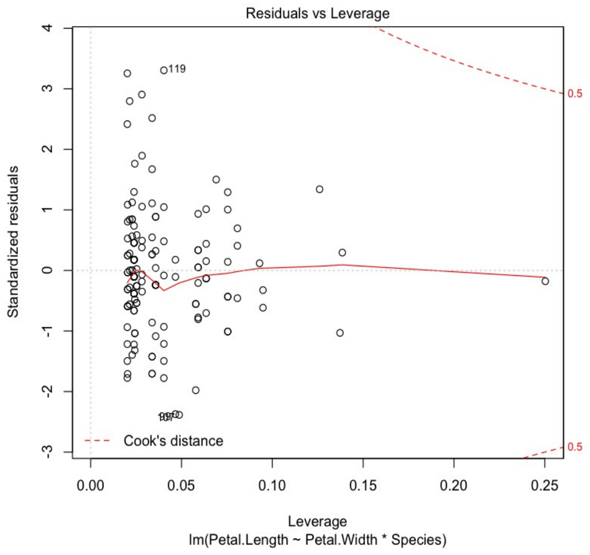
First, we will have a look to the standardized residuals versus fitted values. Remember that this plot should look like a flat line with homogeneous dispersion across all its range:



It looks good except for a few values in the upper right corner that show higher dispersion than the rest of the residuals. The number indicates the position that these data have in their vector (column) of the data frame. There are no weird trends in this plot, it is mainly flat, so it does not seem that we are missing any predictor variable here.
Now, a QQ plot will tell us if the residuals seem to follow a normal distribution or not. If so, they must be close to the diagonal of the plot:



The same as before, it looks good except for a few data in the right corner, these data are the same. Looking at this plot, I bet these residuals are not normal, but we will test it later. The following plot shows the “leverage” effect of each data and the isoclines of “Cook distance”.The leverage shows how far is one data from the average predictor values. The ideal situation is that all the data had low “leverage” effect. As you could imagine, looking at the leverage helps to detect outliers. The magnitude called “Cook distance” tells us how much one specific data is influencing the model fit. We do not want data with high values of Cook distance because, ideally, all of our data should contribute the same to our model. There is no specific limit between “good” and “bad” Cook distance. We just do not want a big dispersion. Outliers also have usually relatively high values of Cook distance.



As we can see, very few data have higher leverage, perhaps only one. But this one is not among those data the showed problems in the two previous plots of normality and residual dispersion. Regarding Cook distance, the outlier has relatively high Cook distance, but also the data that showed problems in the previous plots are close to the isocline of 0.5. As a conclusion, it seems that we will have problems with normality and homoscedasticity,

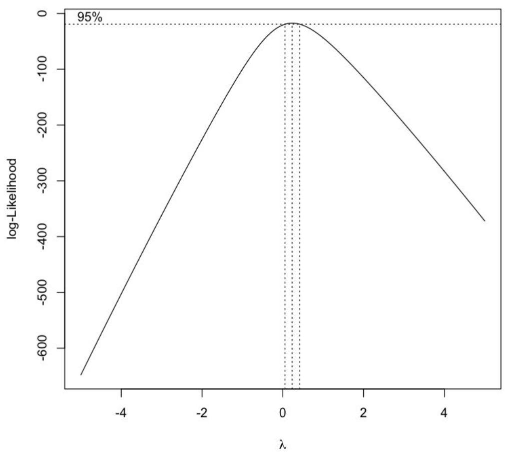
but because both are due to the same data, if we can fix normality, we will probably fix th homoscedasticity problem as well. I would say there is one single outlier, but does not seem to be a problem here.
Removing outliers from a data set is something that could be done when manipulating data, but we must have good reasons to do it. Here we there are no such good reasons.
Now we are going to start applying direct tests for our assumptions. First, the most important, the normality test:

```
> res<-residuals(mod1)
> shapiro.test(res)

        Shapiro-Wilk normality test

data:  res
W = 0.9585, p-value = 0.0001771
```

Residuals do not follow Normal distribution, so we have to fix this problem. And as we know, it is not a problem due to the presence of outliers. Then, we have to try at transforming the variable. You might have heard about log transformation of data, power transformation, root transformation... we might try to apply each of these transformations separately to our data, and see which yields a normal distribution of the residuals. However, there is a more efficient way to proceed. This consists in using the Box-Cox function. This function is actually a family of transformations whose form depends on the value of a parameter called λ. There is a value of λ that optimizes the transformation of our variable in order to get Normal distribution. The goal is to find this value of the parameter. If λ = 0, then the transformation to apply would be Ln(Y). other values of λ equal other kind of transformations. So the Box-Cox function contains most of the possible transformations in one single expression, from which you have to find the optimal. The only problem is that the response variable has to be greater than 0. But this can be solved by rescaling the variable with the addition of a single integer number that makes all the values of Y greater than 0.
we are going to find the optimal value of λ for our response variable. We will do it graphically, which is usually enough. First, we load the package MASS, which contains the function boxcox(). This function gives us the likelihood that certain values of λ will normalize our data. Obviously we want the value of λ with highest likelihood. Then, we apply the function to the model object. The second and third arguments are the range of λ over which to calculate its likelihood, and the fourth argument is the step size of the sequence:

```
> require(MASS)
> likelyfunct<-boxcox(mod1,lambda=seq(-5,5,0.01))
```

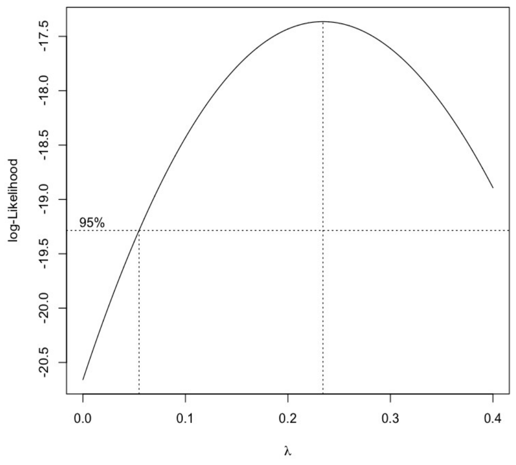
First, we should better have a general overview of the function, in order to locate its maximum. Then we will get closer to the maximum value. So we start with a wide range of λ, from -5 to 5, so that we clearly locate the maximum. We also use a large step size, (0.01) because we do not need accuracy now, and small step sizes take more time for the calculations. This is plot that we obtain:



The optimal value of λ lies somewhere between 0 and 0.4. So we are going to narrow the range of λ now and increase the accuracy of the step size:

```
> likelyfunct<-boxcox(mod1,lambda=seq(0,0.4,0.0001))
```

This is the plot we obtain:



So we can now proceed with the following code, that simply will extract the optimal value of λ (that we will call “lambdaT”):

```
> functX<-likelyfunct$x
> functY<-likelyfunct$y
> maxY<-max(functY)
> XmaxY<-which(functY==maxY)
> lambdaT<-functX[XmaxY]
```

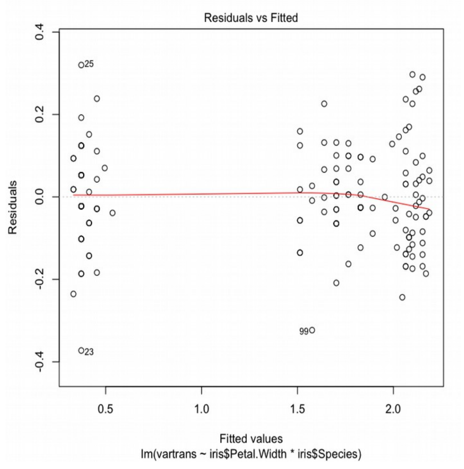
Now, we apply the Box – Cox formula with the “lambdaT”. And we obtain the new, transformed, variable, that we will call “vartrans”:

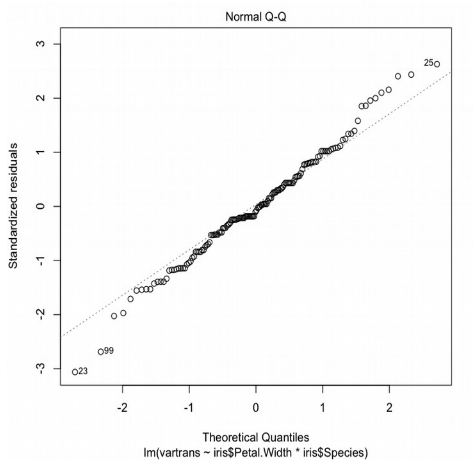
```
> vartrans<-(((iris$Petal.Length)^lambdaT)-1)/(lambdaT)
```

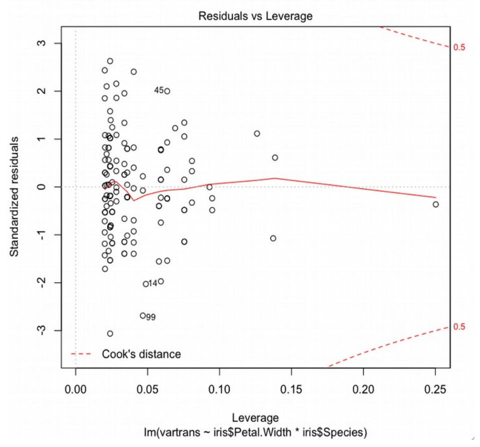
Now, we fit a new model with this transformed variable:

```
> mod2<-lm(vartrans~Petal.Width*Species,data=iris)
> summary(mod2)

Call:
lm(formula = vartrans ~ Petal.Width * Species)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.37258 -0.06459 -0.01056  0.06980  0.32002 

Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)    
(Intercept)                          0.29105    0.04463   6.522 1.10e-09 ***
Petal.Width                          0.40765    0.16701   2.441   0.0159 *  
Speciesversicolor                    0.59021    0.12737   4.634 7.96e-06 ***
Speciesvirginica                     1.45636    0.13839  10.524  < 2e-16 ***
Petal.Width:iris$Speciesversicolor  0.22400    0.18924   1.184   0.2385    
Petal.Width:iris$Speciesvirginica  -0.23170    0.17888  -1.295   0.1973    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1232 on 144 degrees of freedom
Multiple R-squared:  0.9739,     Adjusted R-squared:  0.973 
F-statistic:  1076 on 5 and 144 DF,  p-value: < 2.2e-16
```

Things have changed a bit with this new model. In order to interpret this output, proceed as we did above.
Now we can check the same plots as before. See how they look better now:







Now, we are going to apply our usual Normality test:

```
> res2<-residuals(mod2)
> shapiro.test(res2)

	Shapiro-Wilk normality test

data:  res2
W = 0.9907, p-value = 0.4322
```

Residuals do follow a normal distribution now. So it is time to continue checking the assumptions. Now, the homogeneity of variance among groups. This is performed with Levene's test. There is a function for this test in the *car* package:

```
> require(car)
> leveneTest(res2,iris$Species)
Levene's Test for Homogeneity of Variance (center = median)
       Df F value  Pr(>F)  
group   2  2.6851 0.07156 .
      147                  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The variances are homogeneous. So we are fine with this model. Now we can go the Anova, in order to infer if our predictors have a significant effect.
First, a brief overview of the foundations of the analysis of variance (Anova). This analysis can be applied to normal and homoscedastic data. As a protocol, it has been established that we need at least 30 data, so that our assumptions of Normality etc. could be taken seriously. What this test does is the calculation a ratio of two variances, which is the contrast statistic called *F* (from Fischer). This is its formula:

$$F = \frac{variance\ between\ groups}{variance\ within\ groups} = \frac{RSS_{Treatments} / df\ Treatments}{RSS_{Error} / df\ Error}$$

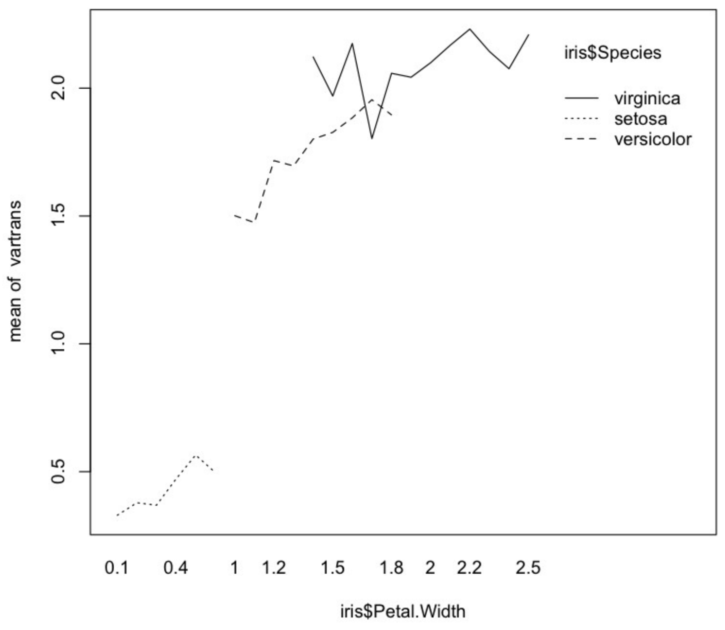
Where *RSS* are the residual sum of squares, and *df* are degrees of freedom.
This formula relies on the partition of the total variance into two main groups (in the simplest case) the *between – groups* variance, defined by the treatment combinations ($RSS_{Treatments} = \sum(y_i - \bar{y})^2$, where y_i is the average of each group or combination of factors, and $\bar{y}$ is the total average), which is obviously due to the treatment effect. And, the *within – groups* variance or residual variance, inside those previous groups, due to differences between individual replicates ($RSS_{Error} = \sum(y_{ij} - y_i)^2$, where y_{ij} is the value of y for each individual replicate, and y_i is again the average of each group or combination of factors). This *within – groups* variance is explained by sampling error or actual differences between individuals). These two RSS, for the *F* formula, have to be scaled relative to the number of elements that they apply to (*df* of treatments, that scales for the free dimensions from the total number of groups, and the *df* of Error, or Residual, that scales for the total number of free dimensions in the residuals, from the total number of individual replicates).
This contrast statistic follows its own probability distribution (*F* distribution) under the null hypothesis that there are no differences between groups. No differences between groups

mean that the variance between groups is the same as the variance within groups. So small values of *F* mean small differences between groups and high values of *F* mean large differences between groups. The *F* probability distribution has low probabilities for the larger values of *F*. So the larger the value of *F*, the larger the probability of finding significant differences. This is how an Anova applied to our model result:

```
> Anova(mod2)
Anova Table (Type II tests)

Response: vartrans
                         Sum Sq  Df  F value    Pr(>F)    
Petal.Width              0.7044   1  46.4106 2.421e-10 ***
Species                  5.1220   2 168.7298 < 2.2e-16 ***
Petal.Width:iris$Species 0.2649   2   8.7277 0.0002644 ***
Residuals                2.1857 144                       
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

All the factors in the model are strongly significant, so we should not eliminate any of them. What this means is that petal width and species can predict petal length. The problem is that the interaction is also significant, so it is not possible to analyze the effect of each variable separately. Instead, we should better have a look of the interaction plots. In this case is simple because the interactions are of second order only, so we need a two – way interaction plot:

```
> interaction.plot(iris$Petal.Width,iris$Species,vartrans)
```



If there was no interaction, all the lines showing the change in petal length (vartrans) from the species *versicolor* to *virginica* would be approximately parallel, because the predictor “species” would have the same effect in petal length (either increasing or decreasing its value). In that case, the effects would be additive. But if the lines had remarkably different slopes, then, it means that the effect is not additive, hence, there is an interaction. Here we can see how the slopes of the lines do not seem to be parallel. When the lines have slopes of different sign, this undoubtedly shows an interaction. In our data, mainly, belonging to the species *virginica* means having larger petal length irrespective of petal width. But there is one clear exception, with the opposite trend for one (or several, we can not see well) value of petal width. Then, we can not say: “the effect of species is positive from versicolor to virginica” because it depends on the value of petal width. Regarding our other predictor, we can say, that petal width has, individually, a positive effect on length. But this effect is different and not proportional between species. The slope of the positive effect of petal width is not the same between the levels of the other factor. So we can say “the effect of petal width is positive” but not “the effect of petal width is homogeneous”.

What does all this mean in conclusion? It means that we are limited regarding our conclusions. Some statisticians would say that a covariate that interacts with a factor should not be included in a model because the slopes between factor groups are not homogeneous. In addition, we can not conclude about the single effect effect of the factor “species” because it depends on the covariate. So, we may say that we do not have any conclusion from our model, except that both predictor variables have influence in the response variable, and that the model can predict the response variable significantly.

It is very common, in linear models, to test specifically the differences between pair – wise comparisons of factor levels. Because we might be interested in knowing which levels have significantly smaller or larger values of the response variable. This are known as “Post – hoc comparisons”. They could be performed with different tests, depending on your needs. For instance, there is the Dunnett test, that compares each factor level against a single control group, or the Tukey test, that makes all the possible pair – wise comparisons. We are going to see an example with our data. Although in our data does not have so much sense to compare the factor levels, from the point of view of the interpretation, because of the presence of the interaction. There are several functions that can perform this task in R. We are going to use glht() (from general linear hypothesis testing) from the package *multcomp*, because it can be applied to objects of many different classes:

```
> require(multcomp)
> summary(glht(mod2,linfct=mcp(Species="Tukey")))

	 Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = vartrans ~ Petal.Width * Species, data = iris)

Linear Hypotheses:
                            Estimate Std. Error t value Pr(>|t|)    
versicolor - setosa == 0      0.5902     0.1274   4.634 1.75e-05 ***
virginica - setosa == 0       1.4564     0.1384  10.524  < 1e-05 ***
virginica - versicolor == 0   0.8662     0.1772   4.889  < 1e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Adjusted p values reported -- single-step method)
```

```
Mensajes de aviso perdidos
In mcp2matrix(model, linfct = linfct) :
  covariate interactions found -- default contrast might be inappropriate
```

The test has found significant differences between all these groups. The estimate of the difference is positive for *versicolor* – *setosa*, so *versicolor* has significantly greater petal length than *setosa*. However, *virginica* has significantly higher average petal length than both *setosa* and *versicolor*.

*Now, just a short note about how to save the diagnostic plots of the model, or any plot that comes in sequential windows. It is easy, just remember from the introduction the sequence *function-for-the-format("file-name.extension") → code of the plot → dev.off()*

```
#Model
> mod<-lm(Petal.Length~Petal.Width*Species,data=iris)

#One way to do it
> pdf("DiagPlot1.pdf")
> par(mfrow=c(2,2))
> plot(mod)
> dev.off()

#Other way to do it
> pdf("DiagPlot2.pdf")
> plot(mod)
> dev.off()
```

"Easier" example **(file R.S3.2)**

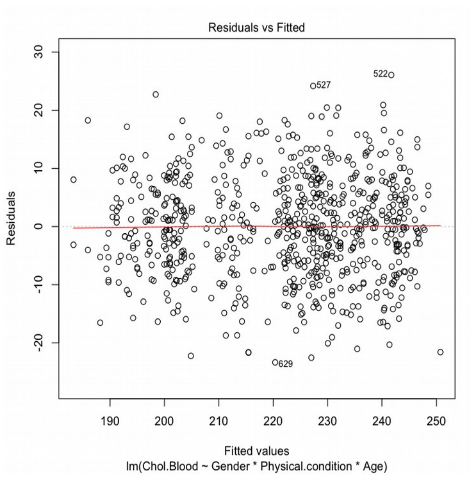
In the previous example, the model gets complicated to interpret because of the interaction between the covariate and the factor. Interactions complicate the interpretation and treatment of models, but they are usually very interesting from the point of view of research. In the next example, we are going to work with a model in which things go smoother. What we have is a dependent or response variable which is the mg per dL of Cholesterol in blood measured in 720 individuals. These individuals were classified according to their age, gender and physical condition (into four *ad hoc* groups).
First, we load the data set:

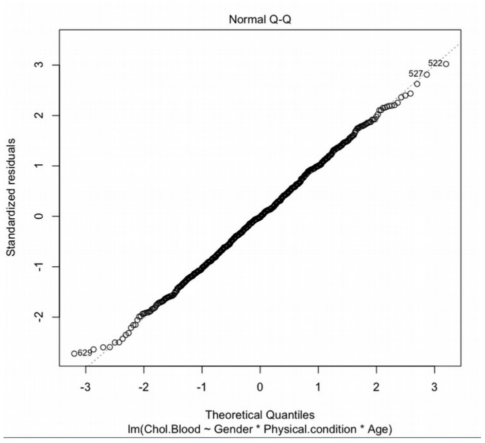
```
> data<-read.csv2("Cholesterol.csv",sep=";")
> head(data)
  X Chol.Blood Age Gender Physical.condition Hospital District
1 1   214.8251  57      M                  A       H1    Porto
2 2   203.5490  45      F                  A       H1    Porto
3 3   221.5127  68      F                  A       H1    Porto
4 4   195.1060  43      F                  A       H1    Porto
5 5   193.9155  36      F                  A       H1    Porto
6 6   200.1424  34      F                  A       H1    Porto
```

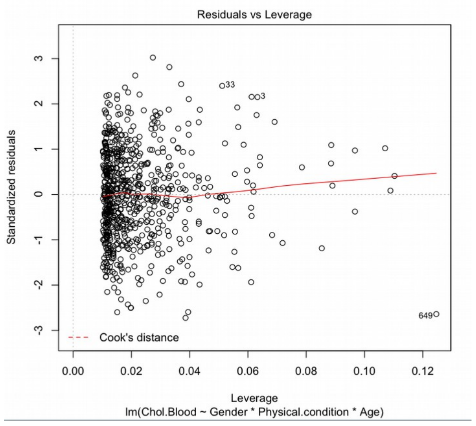
Now we code a linear model, in principle, considering all possible interactions. We have a covariate (Age) and the rest are factors structured in 2 (gender) and 4 (Physical condition) levels:

```
> mod1<-lm(Chol.Blood~Gender*Physical.condition*Age,data=data)
> summary(mod1)

Call:
lm(formula = Chol.Blood ~ Gender * Physical.condition * Age,
    data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-23.353  -5.911  -0.152   5.780  26.059

Coefficients:
                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                     177.613021   3.954843  44.910  < 2e-16 ***
GenderM                          18.060453   5.708270   3.164 0.001624 **
Physical.conditionB              22.115571   5.905822   3.745 0.000195 ***
Physical.conditionC              42.172410   5.762086   7.319 6.84e-13 ***
Physical.conditionD              37.389494   5.957568   6.276 6.07e-10 ***
Age                               0.378012   0.085866   4.402 1.24e-05 ***
GenderM:Physical.conditionB      -3.009284   8.435698  -0.357 0.721399
GenderM:Physical.conditionC     -15.334081   8.227008  -1.864 0.062756 .
GenderM:Physical.conditionD       0.891920   8.334936   0.107 0.914812
GenderM:Age                      -0.224836   0.125463  -1.792 0.073553 .
Physical.conditionB:Age          -0.081420   0.128707  -0.633 0.527201
Physical.conditionC:Age          -0.187814   0.125593  -1.495 0.135253
Physical.conditionD:Age           0.007665   0.128890   0.059 0.952594
GenderM:Physical.conditionB:Age   0.120622   0.185887   0.649 0.516615
GenderM:Physical.conditionC:Age   0.371351   0.179752   2.066 0.039203 *
GenderM:Physical.conditionD:Age   0.062962   0.183417   0.343 0.731498
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 8.745 on 704 degrees of freedom
Multiple R-squared:  0.7816,     Adjusted R-squared:  0.7769
F-statistic: 167.9 on 15 and 704 DF,  p-value: < 2.2e-16
```

Looks like covariate and factors are significant, and less the interaction coefficients, but we will confirm that with an Anova. We can check now the validity of the model with the plots and tests:







The plots look very good in this case. Now, let's proceed with the tests for Normality and homoscedasticity:

```
> res<-residuals(mod1)
> shapiro.test(res)

	Shapiro-Wilk normality test

data:  res
W = 0.9984, p-value = 0.765
```

Residuals are normal.

```
> require(car)
> leveneTest(Chol.Blood~Gender*Physical.condition,data=data)
Levene's Test for Homogeneity of Variance (center = median)
       Df F value Pr(>F)
group   7  1.5445 0.1491
      712
```

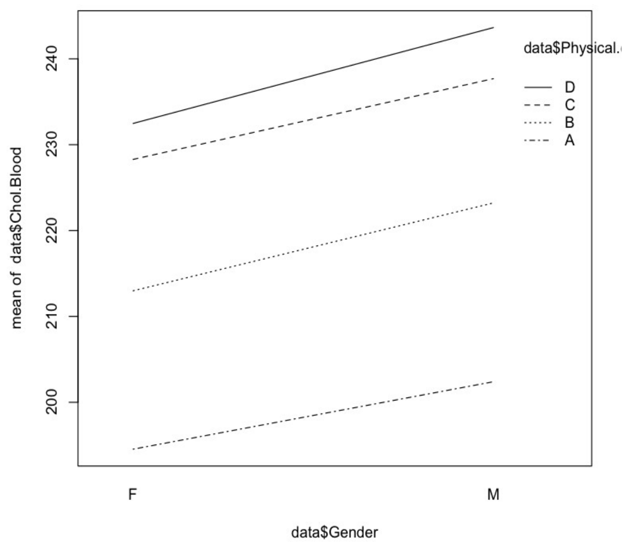
The variances are homoscedastic among the levels of the two factors. Note that in this case, because we have two factors, we have to “cross” the levels of the two factors, so each group over which we are testing the homogeneity of variances is each combination of

the two factors.
So, we can proceed with the Anova:

```
> Anova(mod1)
Anova Table (Type II tests)

Response: Chol.Blood
                             Sum Sq  Df  F value    Pr(>F)    
Gender                        17531   1 229.2511 < 2.2e-16 ***
Physical.condition           170340   3 742.5210 < 2.2e-16 ***
Age                            5258   1  68.7545 5.652e-16 ***
Gender:Physical.condition       328   3   1.4317    0.2323    
Gender:Age                      132   1   1.7216    0.1899    
Physical.condition:Age           30   3   0.1308    0.9418    
Gender:Physical.condition:Age   369   3   1.6088    0.1860    
Residuals                     53834 704                       
---
Signif. Codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

None of the interactions is significant, either of second or third order. In the two-ways interaction plots, we can check how the lines of the effects are parallel. We are only showing here the plot for the Gender:Physical.condition interaction, since it is the easiest to see:

```
> interaction.plot(data$Gender,data$Physical.condition,data$Chol.Blood)
```



So we should better reformulate the model with only additive effects:

```
> mod2<-lm(Chol.Blood~Gender+Physical.condition+Age,data=data)
> summary(mod2)

Call:
lm(formula = Chol.Blood ~ Gender + Physical.condition + Age,
    data = data)
```

```
Residuals:
     Min       1Q   Median       3Q      Max 
-25.0042  -5.9956   0.0571   6.0859  26.5351 

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)         181.55458    1.63698  110.91  < 2e-16 ***
GenderM               9.89249    0.65380   15.13  < 2e-16 ***
Physical.conditionB  19.66610    0.92254   21.32  < 2e-16 ***
Physical.conditionC  34.40663    0.92324   37.27  < 2e-16 ***
Physical.conditionD  39.65073    0.92257   42.98  < 2e-16 ***
Age                   0.26973    0.03273    8.24 8.23e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 8.752 on 714 degrees of freedom
Multiple R-squared:  0.7781,     Adjusted R-squared:  0.7765 
F-statistic: 500.7 on 5 and 714 DF,  p-value: < 2.2e-16
```

So we keep only the most significant model terms. The plots would not change a lot this time. We repeat the Anova with this new model:

```
> Anova(mod2)
Anova Table (Type II tests)

Response: Chol.Blood
                   Sum Sq  Df F value    Pr(>F)    
Gender              17535   1 228.940 < 2.2e-16 ***
Physical.condition 170382   3 741.519 < 2.2e-16 ***
Age                  5200   1  67.897 8.229e-16 ***
Residuals           54686 714                      
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 
```

All the factors as well as the covariate are strongly significant.

How to test linear independence between predictors

We may use this example now for testing dependence between predictors. As we mentioned in the assumptions for general linear models, predictors should be linearly independent. This is also true for multiple regression. This linear independence could be tested simply with pair-wise correlations. But it is often performed with the Variance Inflation factor (VIF). VIF is an index that estimates how much the variance explained is increased by multicollinearity:

```
> require(car)
> vif(mod2)
                       GVIF Df GVIF^(1/(2*Df))
Gender             1.004570  1        1.002283
Physical.condition 1.003354  3        1.000558
Age                1.002458  1        1.001228
```

The VIF values are very low, so we d not should worry about multicolinearity in this case. There is no specific upper limit for the value of VIF, above which we should exclude a variable. Values above 4 – 5 are usually considered problematic.

How to simplify models with a step method

All our predictors have highly significant effects according to the Anova. However, when we have a model with several predictor factors, it is often good to perform some sort of test that tells us if some predictors do not significantly improve the fitting of the model. We can simply do it comparing the AIC values (see the example of non – linear regression) of models starting without factors (only the intercept) and adding one factor each time. The model with significantly lower AIC is the best. Some functions perform this process automatically:

```
> step(mod2)
Start:  AIC=3129.69
Chol.Blood ~ Gender + Physical.condition + Age

                     Df Sum of Sq    RSS    AIC
<none>                             54686 3129.7
- Age                 1      5200  59887 3193.1
- Gender              1     17535  72221 3327.9
- Physical.condition  3    170382 225069 4142.3

Call:
lm(formula = Chol.Blood ~ Gender + Physical.condition + Age,
    data = data)

Coefficients:
        (Intercept)             GenderM  Physical.conditionB  Physical.conditionC
           181.5546              9.8925              19.6661              34.4066
Physical.conditionD                 Age
            39.6507              0.2697
```

This function, step(), in this case by default used a backward method (this default can be modified, of course). So it starts with a full model, and then eliminates step by step, all other predictors. The best model in terms of AIC is still that one with all the predictors. Among the predictors, the one that yields a larger drop in AIC when removed is the Physical condition. So this is probably the most important predictor.
*It is important to say here that the AIC is calculated here a bit in a different way than with the function AIC(), so the values are not the same.

Post – hoc tests

Now that we have our "right" model ready, we might be interested in performing a post – hoc test in order to test for differences between factor levels. We are going to use the same functions as in the previous example:

```
> require(multcomp)
> summary(glht(mod2,linfct=mcp(Gender="Tukey")))

	 Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = Chol.Blood ~ Gender + Physical.condition + Age, 
    data = data)

Linear Hypotheses:
           Estimate Std. Error t value Pr(>|t|)    
M - F == 0   9.8925     0.6538   15.13   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Adjusted p values reported -- single-step method)

> summary(glht(mod2,linfct=mcp(Physical.condition="Tukey")))

	 Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = Chol.Blood ~ Gender + Physical.condition + Age, 
    data = data)

Linear Hypotheses:
           Estimate Std. Error t value Pr(>|t|)    
B - A == 0  19.6661     0.9225  21.317   <1e-07 ***
C - A == 0  34.4066     0.9232  37.267   <1e-07 ***
D - A == 0  39.6507     0.9226  42.978   <1e-07 ***
C - B == 0  14.7405     0.9235  15.961   <1e-07 ***
D - B == 0  19.9846     0.9225  21.663   <1e-07 ***
D - C == 0   5.2441     0.9237   5.677   <1e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Adjusted p values reported -- single-step method)
```

All the pair – wise comparisons between factor levels resulted with significantly different values.

3.2 General Linear model II

Mixed effects models **(file R.S3.3)**

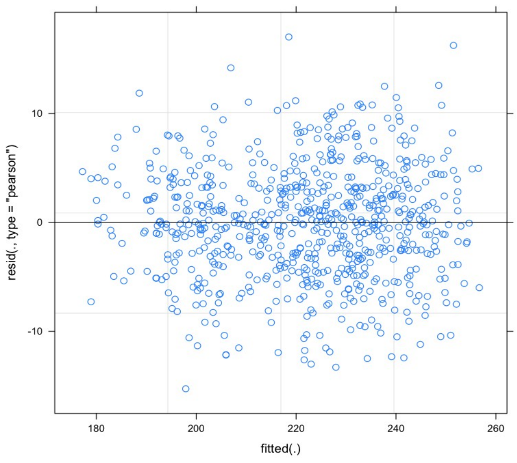
Now, we are going to show a new feature of general linear models: the possibility of including random factors in the analysis. What we have been working with so far were fixed factors. These factors are those that include all the possible levels of a factor, like Gender in our previous example. Or, those that are defined experimentally by the researcher. Random factors are those whose levels in our experiments do not represent all the existing levels, or those that might be affecting our data in a way that we can not control, generating correlation structures within their levels.
The aim of including a random factor in a model is not testing any hypothesis about its effect. It is simply getting rid of the variance due to the autocorrelation within its levels. So they are treated differently than the fixed factors. Using at the same time fixed and random factors is the reason why these models are often called “mixed effects models”.
Previous techniques such as the “repeated measures Anova” are now included in the frame of the General Linear Model as a kind of mixed effects model.
In order to see an example, we are going to use the same data set as in the previous one. And, we are going to include the random factor “Hospital” in the analysis. “Hospital” could be a random factor because it can produce autocorrelation in the values of the response variable that were taken in the same hospital. We do not know it a priori, but we can test it by including this random factor in the model. As you will understand, the hospitals that are included in this analysis do not represent all the possible hospitals. So in principle, it is a good candidate for a random factor.
There are several ways that random factors could be coded in R models. The most common are performed with the function aov() from package *stats*, lme() from the package *nlme*, and lmer() from the package *lme4*. In our example, we are going to use the function lmer():

```
> require(lme4)
> mod3<-lmer(Chol.Blood~Gender+Physical.condition+Age+(1|Hospital),data=data)
> summary(mod3)
Linear mixed model fit by REML ['lmerMod']
Formula: Chol.Blood ~ Gender + Physical.condition + Age + (1 | Hospital)
   Data: data

REML criterion at convergence: 4473.4

Scaled residuals: 
    Min      1Q  Median      3Q     Max 
-2.9306 -0.6296  0.0022  0.6249  3.2678 

Random effects:
 Groups   Name        Variance Std.Dev.
 Hospital (Intercept) 53.52    7.316   
 Residual             27.16    5.211   
Number of obs: 720, groups: Hospital, 12

Fixed effects:
                     Estimate Std. Error t value
(Intercept)         181.05425    2.32788   77.78
```

```
GenderM                9.62741    0.39206   24.56
Physical.conditionB   19.66976    0.54936   35.81
Physical.conditionC   34.38542    0.54978   62.54
Physical.conditionD   39.65618    0.54938   72.18
Age                    0.28405    0.01962   14.47

Correlation of Fixed Effects:
            (Intr) GendrM Phys.B Phys.C Phys.D
GenderM     -0.099
Physcl.cndB -0.118 -0.008
Physcl.cndC -0.114  0.035  0.499
Physcl.cndD -0.118 -0.012  0.500  0.499
Age         -0.377  0.036  0.001 -0.019  0.002
```

The output is very similar to that from lm(). This just gives some more information about the correlation between factors (that we guess is not very important, after checking the VIF). But the most relevant new information is related with the random factor. In the “random effects” section appear the variance absorbed or explained by our random effect. The word “intercept” appears because in our coding we included random effects for the intercept only. So the variance explained by the random effects is quite large, larger than the residual. So this is quite a good thing. We can have a look at the mode plot now. With the function lmer(), only the residual vs fitted values plot is shown:

```
> plot(mod3)
```



This plot looks very good, but mod2 also looked good. Let's have a look at the analysis of deviance, which in this case, of mixed effects models, is actually an analysis of deviance (although the function Anova() performs it as well). The deviance estimates the goodness of fit of a model. It is a generalization of the least squares used in Anova for cases where the fit of the model is performed by maximum likelihood.

```
> Anova(mod3)
Analysis of Deviance Table (Type II Wald chisquare tests)

Response: Chol.Blood
                     Chisq Df Pr(>Chisq)
Gender              602.98  1  < 2.2e-16 ***
Physical.condition 6271.25  3  < 2.2e-16 ***
Age                 209.52  1  < 2.2e-16 ***
```

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The conclusion is very similar to that from the analysis of variance for mod2. In this table the random factor does not appear, because we are not testing any hypothesis about the random factor.
Now we have to choose which is the best model for these data, between mod2 and mod3. In other words, if it is worth including a random effect to explain the variance for these data. For this purpose, first, we could check the AIC values of both models:

```
> AIC(mod2)
[1] 5174.957
> AIC(mod3)
[1] 4489.386
```

The AIC for mod3 is much better. But it is better to check if the difference between these values is significant. Because the models are nested, we can do it with a likelihood ratio test. But, unfortunately, the function anova() does not work when comparing objects of class lm and lmer together. So we have to do it “by hand”. We extract the log likelihood of both models, we apply the formula of the D contrast statistic and the we calculate its probability, that follows a chi – square distribution with the difference in model parameters as degrees of freedom:

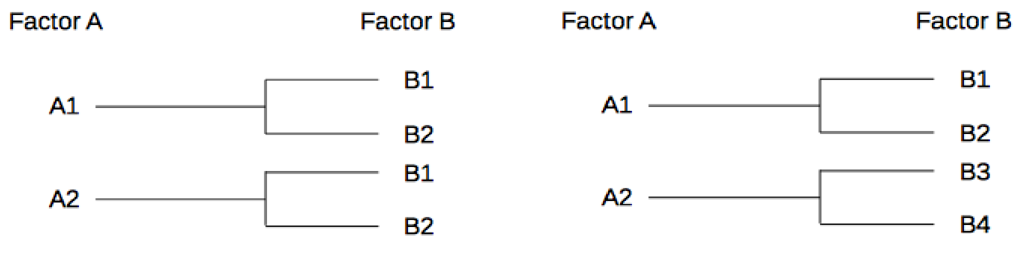
```
> logLik(mod2)
'log Lik.' -2580.479 (df=7)
> logLik(mod3)
'log Lik.' -2236.693 (df=8)
> Dmod2mod3<-(-2*-2580.479)+(2*-2237.039)
> pchisq(Dmod2mod3,df=1,lower.tail=F)
[1] 2.131929e-151
```

The difference is significant, so we keep mod3, with the random factor, as a better model than mod2.

3.3 General linear model III

Nested design (or hierarchical) in the random factor **(file R.S3.3)**

The concept of nested or hierarchical design is important for many cases of linear models. We say that factor 2 is nested into factor 1 when the levels of factor 1 are represented associated with certain levels of factor 1. For instance, in the Cholesterol data from our previous examples, we have the factor "Hospital" with 12 levels. But these 12 levels are associated with another factor, the factor "District". So that levels 1 to 4 are in level "Porto", 5 to 8 in level "Aveiro" and 9 to 12 in level "Braga". We can say then that factor "Hospital" is then nested in the factor "District". If data from the same hospital were distributed through several districts (obviously impossible) then there would be no nested factors.



Factor B is NOT NESTED in factor A   Factor B is NESTED in factor A

Nested factors could exist within fixed or random factors. A statistical test called "nested Anova" is now performed inside the general frame of General Linear Model. They are coded in different ways depending on the function. We are going to see an example with our cholesterol data, where the factor "Hospital" is nested in "District". Be always careful with the notation of nested factors, because, independently of the function, they are always notated from the "broadest" to the "narrowest" reading from the left to the right:

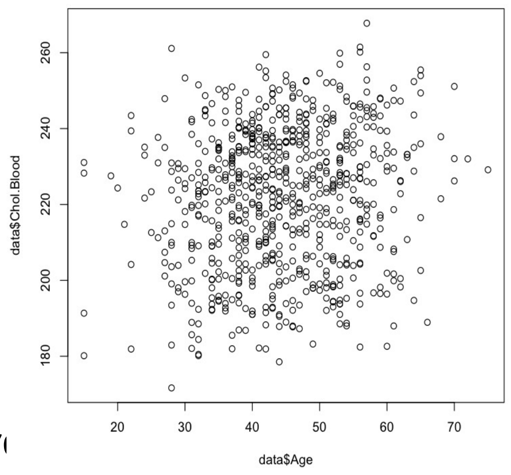
```
> mod4<-lmer(Chol.Blood~Gender+Physical.condition+Age+(1|District/Hospital),data=data)
> summary(mod4)
Linear mixed model fit by REML ['lmerMod']
Formula: Chol.Blood ~ Gender + Physical.condition + Age + (1 | District/Hospital)
   Data: data

REML criterion at convergence: 4473.4

Scaled residuals: 
    Min      1Q  Median      3Q     Max 
-2.9306 -0.6296  0.0022  0.6249  3.2678 

Random effects:
 Groups            Name        Variance  Std.Dev.
 Hospital:District (Intercept) 5.352e+01 7.316e+00
 District          (Intercept) 6.453e-13 8.033e-07
 Residual                      2.716e+01 5.211e+00
Number of obs: 720, groups: Hospital:District, 12; District, 3

Fixed effects:
                     Estimate Std. Error t value
```

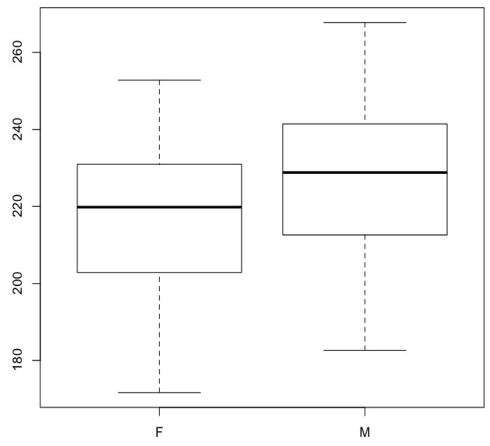
```
(Intercept)         181.05425    2.32788   77.78
GenderM               9.62741    0.39206   24.56
Physical.conditionB  19.66976    0.54936   35.81
Physical.conditionC  34.38542    0.54978   62.54
Physical.conditionD  39.65618    0.54938   72.18
Age                   0.28405    0.01962   14.47

Correlation of Fixed Effects:
            (Intr) GendrM Phys.B Phys.C Phys.D
GenderM     -0.099
Physcl.cndB -0.118 -0.008
Physcl.cndC -0.114  0.035  0.499
Physcl.cndD -0.118 -0.012  0.500  0.499
Age         -0.377  0.036  0.001 -0.019  0.002
> Anova(mod4)
Analysis of Deviance Table (Type II Wald chisquare tests)

Response: Chol.Blood
                     Chisq Df Pr(>Chisq)
Gender              602.98  1  < 2.2e-16 ***
Physical.condition 6271.25  3  < 2.2e-16 ***
Age                 209.52  1  < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

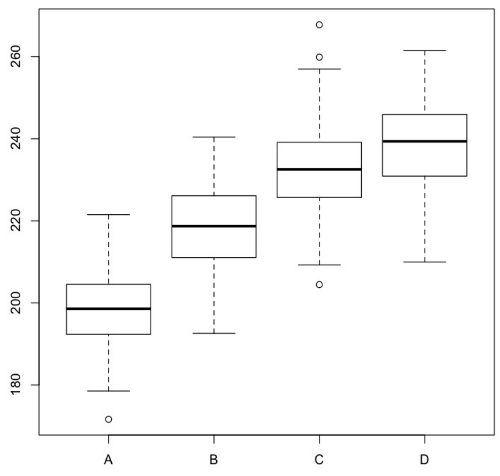
In the output, we see that the variance absorbed by “District” is very low. So it seems that this factor is not adding much to our model. Then, it is time to choose the best model between mod3 and mod4:

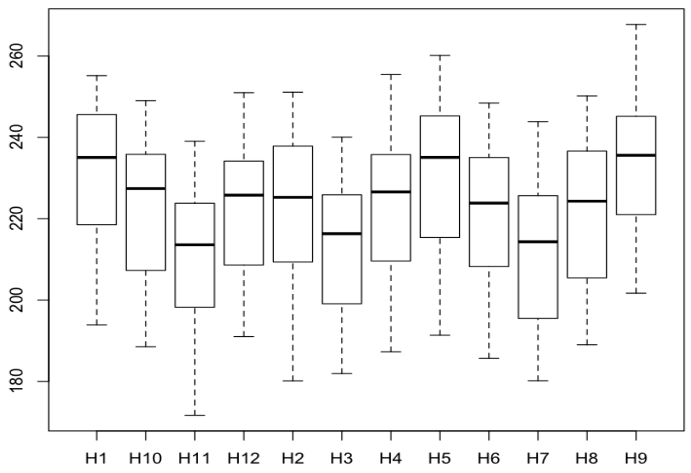
```
> AIC(mod3)
[1] 4489.386
> AIC(mod4)
[1] 4491.386
```

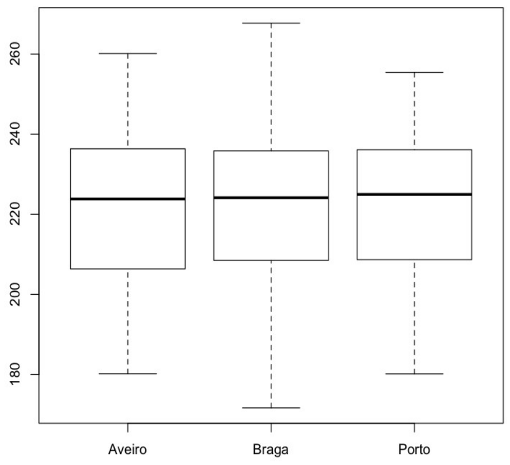
The AIC for mod3 is already lower than for mod4. So we do not even need to perform the log likelihood ratio test, since mod4 has more parameters than mod3. So the increase in variance explained by mod4 with respect to mod3 (6.453^{-13}) that obviously, can not balance the penalty of having one more extra – parameter. So, for the Cholesterol data, our best model is mod3. This model says that only the covariate, the fixed factors, and the random factor “hospital” are significant. Let's have a look in s few plots how the influence of these factors look like:







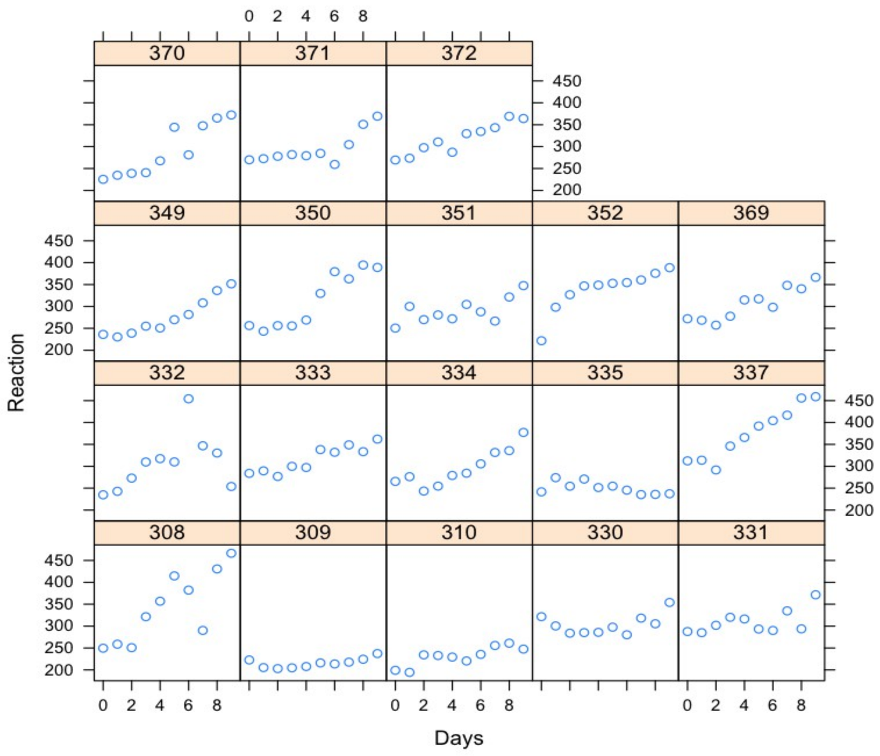




These plots confirm very well the results from our models. All the fixed factors clearly have an effect. The effect of the covariate is, however, difficult to see because we plotted all the data together. The random factor Hospital also has a clear and consistent effect. But, the District where the hospitals are located has no influence at all.

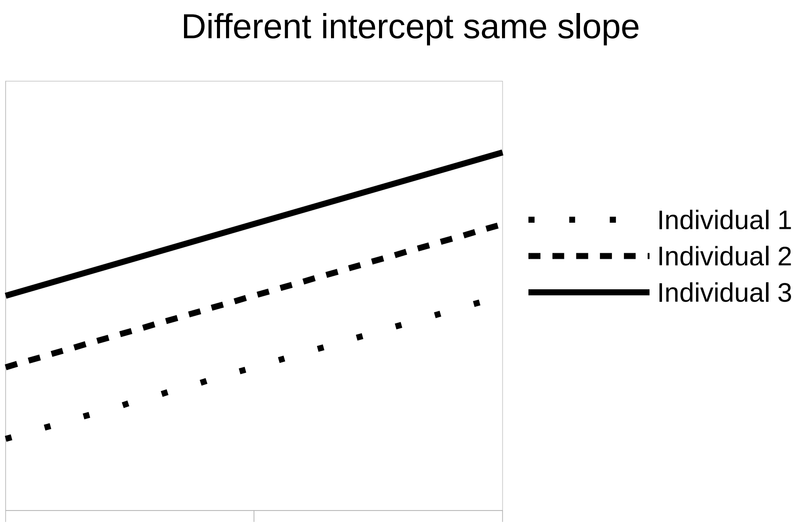
Random slopes **(file R.S3.3)**

Briefly, we are going to analyze data from a data set in the *datasets* package. The data set is named *sleepstudy*. The data consist in the evolution of the reaction time in a group of individuals (soldiers) after been certain time without sleeping. Here is how the data look:



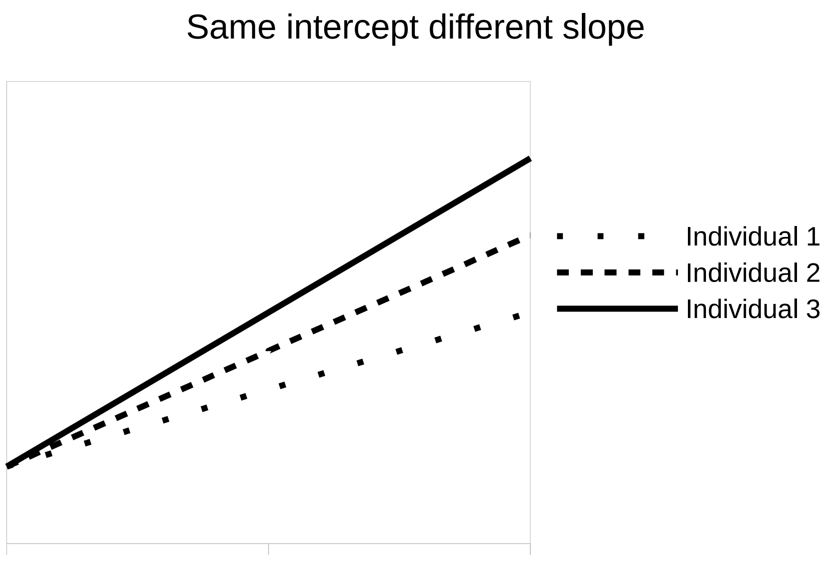
This graphical function xyplot() is a nice function from the library, or graphical package *lattice*, that is installed by default, but not loaded. Here we can see how each individual has its own "slope" for the increase in reaction time after days without sleeping. Then, we can not use a single relationship of reaction time versus days without sleeping, because there is each individual is different, has its own slope of reaction, there is this autocorrelation within individual. This is an example of what could be called "random slopes". But, it is very similar to what is called a "repeated measures design" (old "repeated measures anova"). So the individual is a random factor here.
*By the way, a very common way of calling random factors is "block factor". Here, the "blocks" are the individuals. In our previous example, the blocks were the Hospitals.

In the precious example (Cholesterol) the effect of the random factor could be on the intercept, but not the slope. Here is a conceptual graphic scheme of this kind of model desig, and below, the R code for our current example:



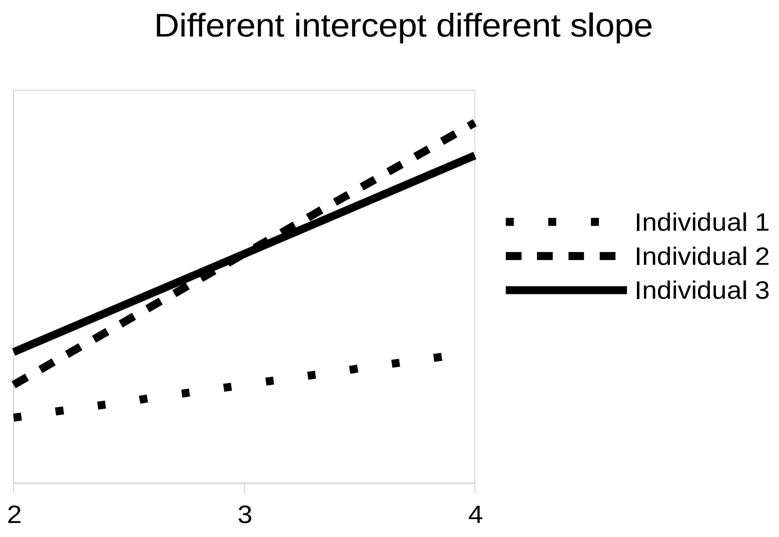
```
> mod1<-lmer(Reaction~Days+(1|Subject),data=sleepstudy)
```

But it does not seem, looking at our data, that we are in that previous situation. However, it I good to test the model anyways, and see how its fits the data compared to other model alternatives.
It does not seem either that we are in a case where the effect is only in the slope but not in the intercept:



```
> mod2<-lmer(Reaction~Days+(0+Days|Subject),data=sleepstudy)
```

It is more likely that our random factor affects both the slope and the intercept:



Intercept and slope correlated:

```
> mod3<-lmer(Reaction~Days+(Days|Subject),data=sleepstudy)
```

Intercept and slope not correlated:

```
> mod4<-lmer(Reaction~Days+(1|Subject)+(0+Days|Subject),data=sleepstudy)
```

Let's see what the AIC tells us about the best model:

```
> AIC(mod1)
[1] 1794.465
> AIC(mod2)
[1] 1774.525
> AIC(mod3)
[1] 1755.628
> AIC(mod4)
[1] 1753.669
```

According to AIC, the best model is mod3. But because the models are nested, we can check with a likelihood ratio test if the differences are significant:

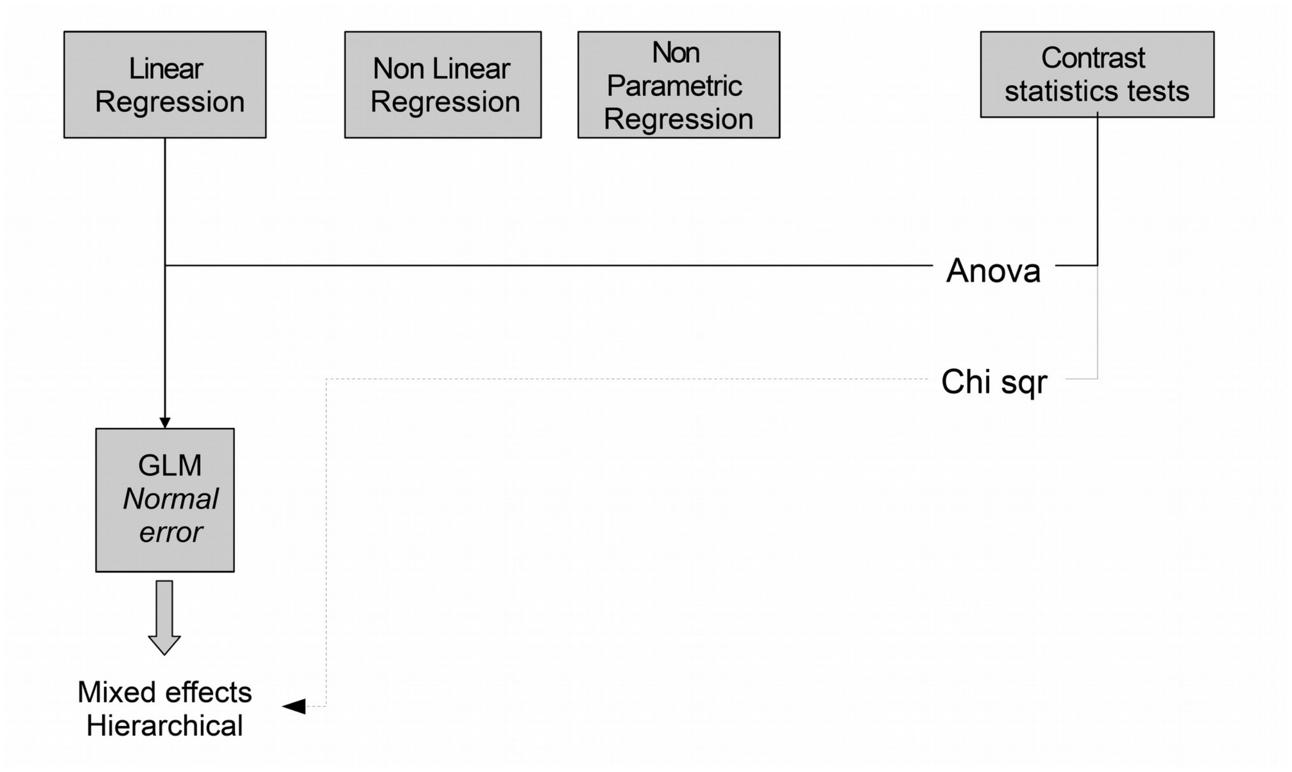
```
> anova(mod1,mod2,mod3,mod4)
refitting model(s) with ML (instead of REML)
Data: sleepstudy
Models:
mod1: Reaction ~ Days + (1 | Subject)
mod2: Reaction ~ Days + (0 + Days | Subject)
mod4: Reaction ~ Days + (1 | Subject) + (0 + Days | Subject)
mod3: Reaction ~ Days + (Days | Subject)
     Df    AIC    BIC  logLik deviance   Chisq Chi Df Pr(>Chisq)
mod1  4 1802.1 1814.8 -897.04   1794.1
mod2  4 1782.1 1794.8 -887.04   1774.1 19.9983      0  < 2.2e-16 ***
mod4  5 1762.0 1778.0 -876.00   1752.0 22.0771      1  2.619e-06 ***
mod3  6 1763.9 1783.1 -875.97   1751.9  0.0639      1     0.8004
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

So the differences between mod1 and mod2 and mod2 and mod4 are significant. So mod1 should not be considered at all. However, the difference between mod3 and mod4 is not significant. Note that the AIC values are a bit different because, as the message in the first line says, the models were refitted using ML (maximum likelihood). We could use the method “ML” in the lmer() function as well. It is just that REML (restricted maximum likelihood) is the default method. But the conclusions would be very similar.
In conclusion, even though AIC favors mod4, we should choose mod3 because the difference in AIC is not significant, and mod2 has one parameter less. This is the way to keep with the principle of maximum parsimony. However, the correlation between the random slope and intercept is very low. Form this point of view, mod4 makes more sense.

```
> summary(mod3)
Linear mixed model fit by REML ['lmerMod']
Formula: Reaction ~ Days + (Days | Subject)
```

```
   Data: sleepstudy

REML criterion at convergence: 1743.6

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.9536 -0.4634  0.0231  0.4634  5.1793

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 Subject  (Intercept) 612.09   24.740
          Days         35.07    5.922   0.07
 Residual             654.94   25.592
Number of obs: 180, groups: Subject, 18

Fixed effects:
            Estimate Std. Error t value
(Intercept)  251.405      6.825   36.84
Days          10.467      1.546    6.77

Correlation of Fixed Effects:
     (Intr)
Days -0.138
```

Now, before going to the next section, let's see a scheme picture of our statistical models:



Linear Regression
Non Linear Regression
Non Parametric Regression
Contrast statistics tests
Anova
Chi sqr
GLM *Normal error*
Mixed effects Hierarchical

Section 4: Introduction to GLZ, GAM and Meta-Analysis

Contents

4.1 Introduction to Generalized Linear Models

These models (GLZ) are a generalization of the GLM to situations where there might be some sort of non linearity (smooth monotonically increasing functions) and the model error, or residuals, do not necessarily follow a Normal distribution. So, in the model, we have to specify which distribution follows the error term. The possible distributions belong to the exponential family (Poisson, Gaussian = Normal, Binomial, Gamma, and other related). It is very common to use these models for errors following binomial distribution. Binomial distributions describe the error of phenomena with only two possible outcomes (like presence/absence). The GLZ with binomial error distribution was previously known (and still often today) e previously known “logistic regression”. The GLZ with Poisson distribution was formerly known as “Poisson regression” or “log normal regression”.
In the GLZ, the relationship between the predictors and the response variable is not defined with a model for the data, as in the GLM. This is because in the GLZ we may need non linear – models. Instead, the relationship is defined with a model function for the relationship between the predictors and expected value of the response variable. What this model function does is linearizing the relationship. This model function is called “link” function. For the Binomial distribution, the link function is the *logit* function. For the Poisson distribution, the link function is the *log* function. For the Normal or Gaussian distribution, the link function is the *identity* function.
These concepts are summarized in the following equations. A GLM can be simplified:

$$\mu_i = ß_i X_i + \epsilon_i$$

Where μ_i are the expected or predicted values of the response variable, $ß_i$ the coefficients of the predictor variables X_i, and ϵ_i are the model errors, which are normally distributed.
Then, if the formula of a GLZ model can be simplified as:

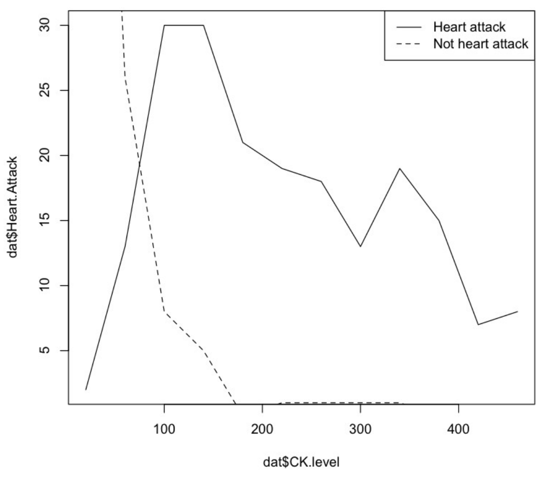
$$g(\mu_i) = ß_i X_i + \epsilon_i$$

Where *g()* is the link function, that linearizes the relationship (without g(), ß could be not linear respect to μ, and ϵ not normally distributed).
Typical variables that are analyzed with GLZ are count data, binomial outcomes like presence/absence and phenomena that show aggregation in the response. They are interesting, although less usual, the “zero inflated models”, which are data that have an excess of zero values, like the count of rare and/or aggregated events. Ther is also a “multinomial” version of logistic regression for those depedent varaibles that have more than 2 categories.
Apart from these features, GLZ are similar in structure to the GLM. They can admit random factors (Generalized mixed linear models, called GLMM) nested factors, etc.
Another difference comes at the evaluation of the main effects, which can not be performed with Anova, since the Fischer – Snedecor *F* statistic is sensitive to non – normality of errors. Instead, it is used the Analysis of Deviance, which consists in the same idea of the residuals least squares estimate of the unexplained variance generalized to models fit by maximum likelihood estimates. The contrast statistic follows a chi – squared distribution.

Example with binomial error distribution **(file R.S4.1)**

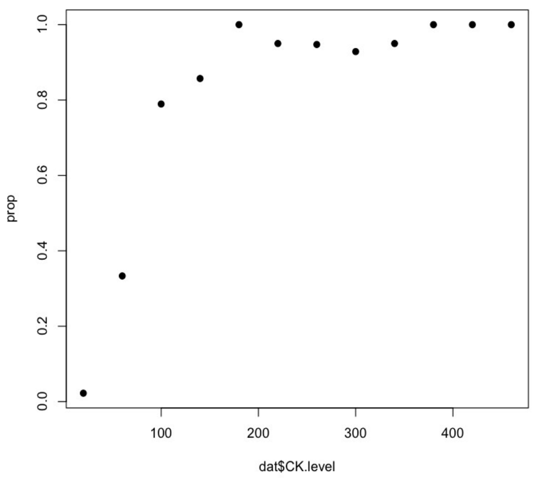
We are going to use an example taken from Simon N. Wood (University of Bath, UK) course on statistics. These data consist in the relation between blood creatine kinase levels and suffering a heart attack (this is a binomial outcome: heart attack/no heart attack). The data on suffering heart attack are the numbers of patients that suffered/not suffered heart attack. This is a plot of creatine kinase levels versus the numbers of patients with the two outcomes.

```
> plot(dat$CK.level,dat$Heart.Attack, type="l")
> points(dat$CK.level,dat$Not.Heart.Attack,type="l",lty=2)
> legend("topright",c("Heart attack","Not heart attack"),lty=c(1,2))
```



And these are the levels of Creatine Kinase versus the proportion of patients suffering a heart attack.

```
> prop<-dat$Heart.Attack/(dat$Heart.Attack+dat$Not.Heart.Attack)
> plot(dat$CK.level,prop,pch=19)
```



As you can guess from these two plots, the creatine kinase levels are a good diagnosis for

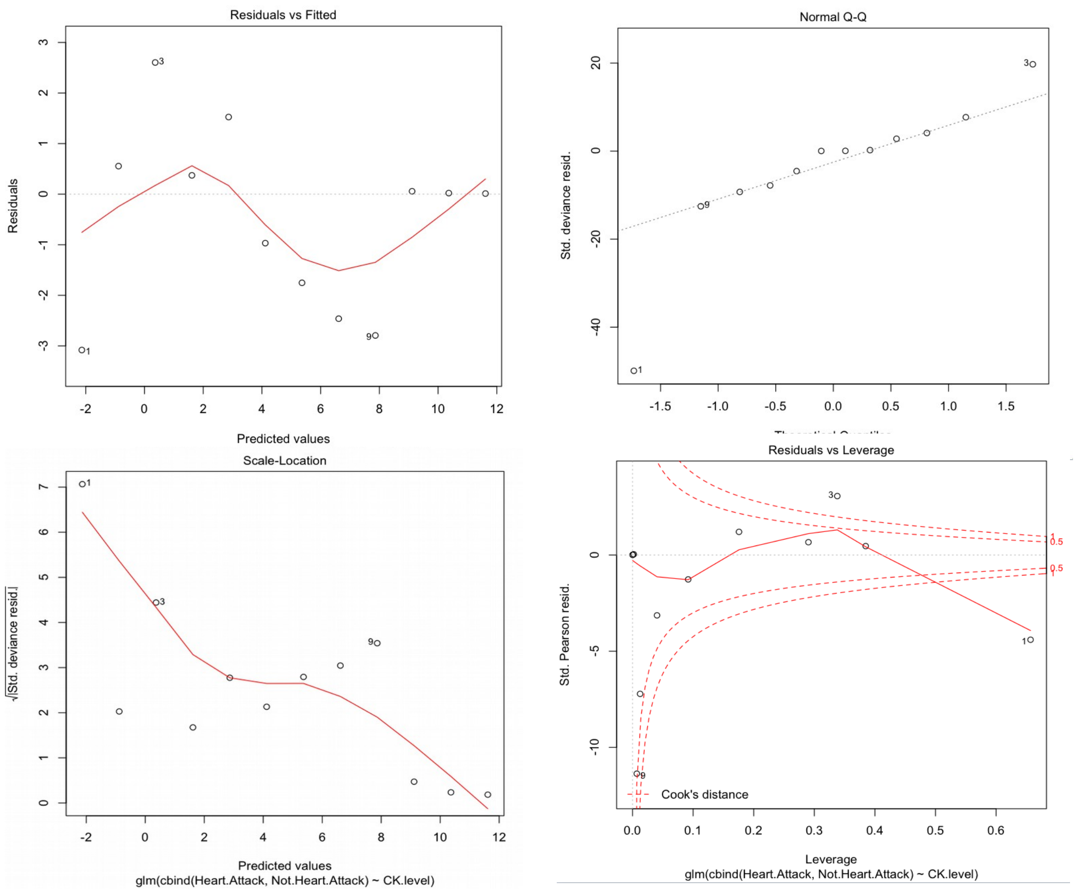
the risk of suffering heart attacks. Because we have a binomial outcome, we need to provide a binomial distribution as an error distribution.
Now, let's fit a model with R. The function for GLZ is glm(). Our response variable can not be just the proportion of heart attacks. This is because this proportion comes from evaluating a different number of patients at each value of creatine kinase, and we need to provide information about this number of patients (*n* of each outcome). This can be done in two different ways:
- Provide the proportion of heart attacks and, with the argument “weights” a vector of weights with the number of patients (*n*) from which each outcome was evaluated.
- Provide the response variable as a two column vector with the number of patients for the two results of the outcome.

This would not be needed if our response variable was a binary variable, where there ir no effect of *n*.
In our present case, we are going to choose the second option:

```
> mod1<-glm(cbind(Heart.Attack,Not.Heart.Attack)~CK.level,family=binomial,data=dat)
> summary(mod1)

Call:
glm(formula = cbind(Heart.Attack, Not.Heart.Attack) ~ CK.level,
    family = binomial, data = dat)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-3.08184  -1.93008   0.01652   0.41772   2.60362

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.758358   0.336696  -8.192 2.56e-16 ***
CK.level     0.031244   0.003619   8.633  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 271.712  on 11  degrees of freedom
Residual deviance:  36.929  on 10  degrees of freedom
AIC: 62.334

Number of Fisher Scoring iterations: 6
```

The information provided in the summary is very similar to that of the linear models. Except that now, the Deviance tells us more or less what the residual squares error in the GLM. Now, we can plot the model to check the behavior of residuals vs predicted values, QQ plot, etc.:

```
> plot(mod1)
```



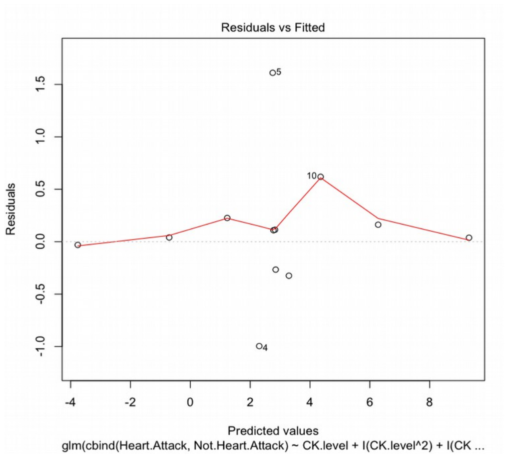
The interpretation of these plots is very much like that for GLM, except that there might be departures from normality in the QQ plot because the predicted values are in the scale of the linear predictor, rather than the response variable (remember that in the GLZ the response variable is not linear against the predictors, it is linearized with the link function). But we see two things that do not look good: first, a strange trend in the residual vs fitted plot, and second, two points (3 and 1) with large values of Cook distance (and also leverage in the case of point 1).
These problems are not easy to solve. One of the reasons is that there are very few data. With few data it is easier that certain data points had large leverage or Cook's distance values. The trend on the residuals vs fitted plot means that we are missing something in the model predictors. “Something” means more parameters, which could be obtained with more predictor variables. But we do not have more predictor variables for our data. So, we could also try a non – linear function, that uses more parameters, without increasing the number of predictor variables. For instance, we could use a cubic function. This is suggested by the “S” shape of the trend in the residual vs fitted values plot. In a GLM model we can not change the functional form from being linear. But, as we said in the introduction to this section, the GLZ allow non linear functions if they are smooth and monotonically increasing. So, this could be a new model formulation to try:

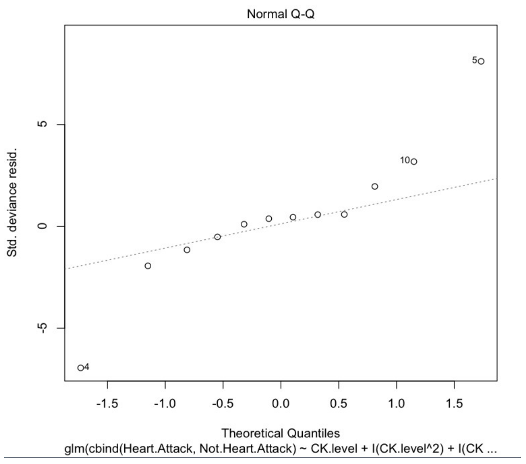
```
> mod2<-
glm(cbind(Heart.Attack,Not.Heart.Attack)~CK.level+I(CK.level^2)+I(CK.level^3),family=bino
mial,data=dat)
> summary(mod2)
```

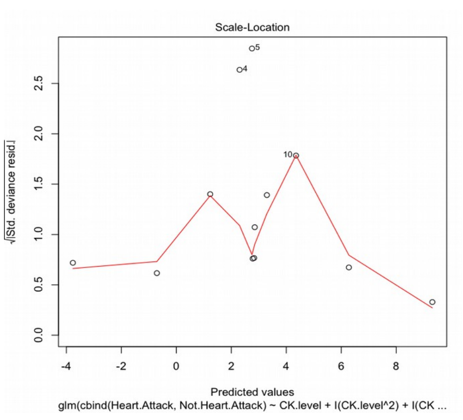
```
Call:
glm(formula = cbind(Heart.Attack, Not.Heart.Attack) ~ CK.level +
    I(CK.level^2) + I(CK.level^3), family = binomial, data = dat)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.99572  -0.08966   0.07468   0.17815   1.61096

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)   -5.786e+00  9.268e-01  -6.243 4.30e-10 ***
CK.level       1.102e-01  2.139e-02   5.153 2.57e-07 ***
I(CK.level^2) -4.649e-04  1.381e-04  -3.367  0.00076 ***
I(CK.level^3)  6.448e-07  2.544e-07   2.535  0.01125 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 271.7124  on 11  degrees of freedom
Residual deviance:   4.2525  on  8  degrees of freedom
AIC: 33.658

Number of Fisher Scoring iterations: 6
```

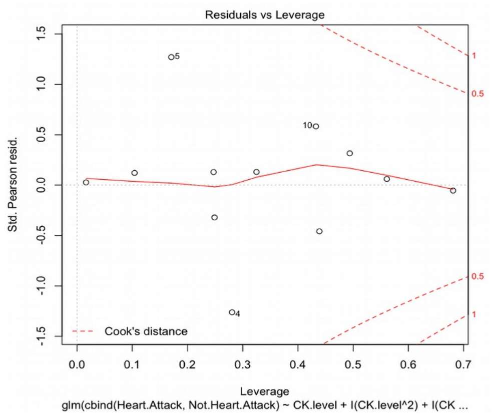
This model looks promising. All its terms are significant, and the residual deviance was reduced from 36.9 to 4.2. This, at the cost of adding two extra – parameters (the coefficients for the squared and the cubic term). We will see later if the increase in the explained deviance could balance the penalty paid for these extra 2 parameters.
Now, let´s have a look to the new model plots:

```
> plot(mod2)
```









A few things look better know. There is no weird trend in the residual vs. fitted plot, and the leverage and Cook distances are in an adequate range. However, there are still a few points that look a bit “out” of the model standards. But, as it was said before, with so few data it is difficult not to have any problem. Now, let's look at the AIC for both models:

```
> AIC(mod1)
[1] 62.3339
> AIC(mod2)
[1] 33.65773
```

The AIC from model 2 looks definitely better. So the penalty paid for having to extra – parameters is balanced by the increase in explanation of deviance. It is probably significantly different, but it is always good to test it with a likelihood – ratio test (since the models are nested):

```
> anova(mod1,mod2,test="Chisq")
Analysis of Deviance Table

Model 1: cbind(Heart.Attack, Not.Heart.Attack) ~ CK.level
Model 2: cbind(Heart.Attack, Not.Heart.Attack) ~ CK.level + I(CK.level^2) +
    I(CK.level^3)
  Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
1        10     36.929
2         8      4.252  2   32.676 8.025e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

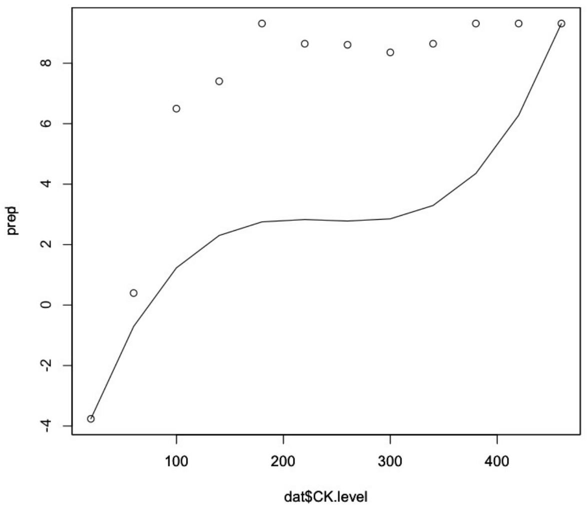
The difference is significantly, so we have to select mod2 as the best model. Let's see now how the analysis of deviance looks:

```
> require(car)
> Anova(mod2)
Analysis of Deviance Table (Type II tests)

Response: cbind(Heart.Attack, Not.Heart.Attack)
              LR Chisq Df Pr(>Chisq)
CK.level        47.863  1  4.570e-12 ***
```

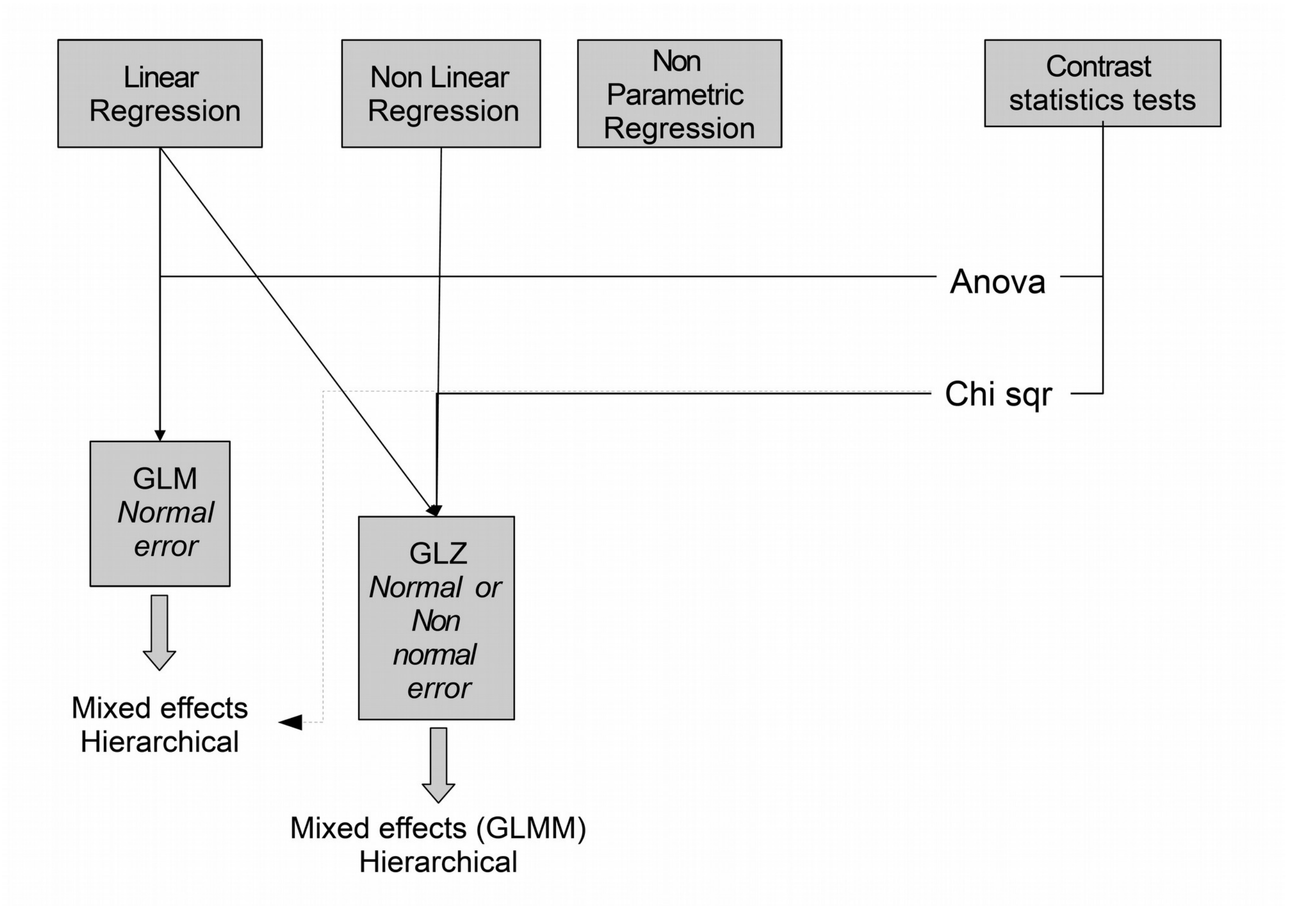
```
I(CK.level^2)   18.027  1  2.178e-05 ***
I(CK.level^3)   11.158  1  0.0008368 ***
---
Signif. Codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The effects are all highly significant.
If needed (this is not our case here) it is possible to perform post – hoc tests, as in the GLM, with the glht() function, from package *multcomp*.
It is interesting to know the predicted probabilities from mod2 for suffering or not heart attack, comparing them in a plot with the actual proportion:

```
> pred<-predict(mod2)
> plot(dat$CK.level,pred,type="l")
> par(new=T)
> plot(dat$CK.level,prop,yaxt="n")
```



Now we are able to add the GLZ component to our scheme from page 81:


Linear Regression
Non Linear Regression
Non Parametric Regression
Contrast statistics tests
Anova
Chi sqr
GLM *Normal error*
GLZ *Normal or Non normal error*
Mixed effects Hierarchical
Mixed effects (GLMM) Hierarchical

4.2 Introduction to Generalized Additive Models

As we have discussed in the section devoted to non – parametric regression (Section 1.4, page 32) the non parametric models, in certain aspects, offer advantages with respect to parametric models. Parametric models have the problem that their success in the fitting process depends a lot on the sample data, particularly when data are few and there is no good knowledge of the underlying processes. So there is a risk of fitting a wrong model to data. Then, parametric models have the advantage of being very stable and producing very accurate predictions. But if you have chosen the wrong model, these advantages become a problem. On the other hand, non – parametric models are more flexible, they can adapt better to data, they can lead better with occasional perturbations in the data. In general they are more sensitive to the factors that are influencing a process, so it is easier to detect the effect of certain variables in your data that the “rigid” parametric forms may not detect.
So, the technique of Generalized Additive Models (GAM) aims to extend the advantages of GLM and GLZ for hypothesis testing to a frame were models are fitted with non parametric regression methods. But, initially, there was a problem called “the curse of dimensionality”. This problem arises when considering the effect of various predictors in a response variable (in our previous examples of non – parametric regressions there was only one predictor, but extending the GLM, we are interested in having more predictors). If each predictor is fitted with a non parametric method, the final models that comes out, considering all parameters and possible interactions, has an enormous complexity in terms of dimensions, that make it impossible to be estimated in the frame of your data unless you have extremely high numbers of data. This problem was solved by simplifying the model, assuming that the effects of the predictor functions are always additive, and so there is not a huge number of interactions between parameters (so here is why they are called generalized “additive” models). Basically, this is a similar idea as the introduction of the “link” function in GLZ. The “link” function in GAM would be then the set of non – parametric functions that predict our response variable. Here is a general formula for these models that exemplifies well this aspects. If the predicted values of our response variable (μ_i) are given by an additive series of non parametric functions (f_i):

$$\mu_i = f_i(X_i) + \ldots + f_n(X_n) + \epsilon_i$$

Using the link function, this can be have the form of a simple additive GLM:

$$g(\mu_i) = \sum_j f_j(X_i) + \epsilon_i$$

Where the sum of f_i functions is linear with respect to g(u_i).
The error term could be any distribution from the exponential family (Normal, Poisson, Binomial, etc.) same as for the GLZ.

Example **(file R.S4.2)**

We are going to work a classic example of GAM from Hastie & Tibshirani (1990), named “kyphosis”. The data set is available in the package *gam*, which, together with package *mgcv* are those that offer the best tools from R for GAM and GAMM modeling. The data consist in the results of a spinal operation called laminectomy on children, to correct for a condition called “kyphosis”. The response variable is binary (presence/absence of kyphosis), and the independent variables are the age of children in months, the number of vertebra involved in the operation, and the level of the operation.

```
> require(gam)
> data(kyphosis)
> head(kyphosis)
  Kyphosis Age Number Start
1   absent  71      3     5
2   absent 158      3    14
3  present 128      4     5
4   absent   2      5     1
5   absent   1      4    15
6   absent   1      2    16
```

Choosing an appropriate GAM model is complicated because of the great variety of model forms available. There is a choice of variables to include in the model (like in GLM or GLZ) but there is a choice as well of error distribution (unlike GLM, but same as GLZ) and the most complicated aspected, that does not exist in GLM and GLZ is the choice of smooth non parametric functions for each predictor. These smooth functions can be obtained with polynomials, splines, etc. and several choices are to be made as well regarding the degree of polynomials, kernel, bandwidth for smoothing (*h*). In order to select the best model form we need the help of model plots and specific selection criteria, similar to those that we employed for the GLM and GLZ.
In our present example, there is at least one thing that is clear: the response or dependent variable has binomial error distribution, since it is a binary variable (presence/absence). Regarding the bandwidth selection criteria, we are going to choose one of the methods that in principle I consider the most objective and widely applicable: the generalized cross – validation. This method is implemented by default in the gam() function from package *mgcv* (do not mistake with function gam() from package *gam*). This function only has the spline method as a method to define the smoothed functions, and it selects automatically as well the degree of smoothing of the splines. So, in this example, we are going to decide only which variables to include in the model (and also which of them to smooth or not). Another difference between these two functions is that the function gam() from *gam* package uses a *backfitting* algorithm for model fit, whereas gam() from *mgcv* uses a penalized splines (likelihood – based) method.
So, if we want to use different methods than the splines for obtaining the smooth functions (polynomial, loess) these are only supported by gam() function from *gam* package.
As we know, from our previous introduction, GAM do not consider interactions between predictors (as GLM or GLZ do) so we do not have to test them in our model. This simplification is an advantage for the analysis of these models, but on the other hand is a limitation for cases when strong interactions actually exist among our predictors. In the coding of these models with R, it is still possible to include interactions, but they are not recommended, unless if we are talking about one or some few low order interactions.
As a starting point it is good to fit a GLZ, in order to compare its fit with the forthcoming GAM models:

```
> mod1<-glm(Kyphosis~Age*Start*Number,family=binomial,data=kyphosis)
> summary(mod1)
```

```
Call:
glm(formula = Kyphosis ~ Age * Start * Number, family = binomial,
    data = kyphosis)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8250  -0.4480  -0.2835  -0.1580   2.2782

Coefficients:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)      -17.672117   8.894370  -1.987   0.0469 *
Age                0.156794   0.071804   2.184   0.0290 *
Start              0.842401   0.650628   1.295   0.1954
Number             3.220739   1.754246   1.836   0.0664 .
Age:Start         -0.010049   0.005469  -1.837   0.0662 .
Age:Number        -0.025389   0.013597  -1.867   0.0619 .
Start:Number      -0.181746   0.135846  -1.338   0.1809
Age:Start:Number   0.001667   0.001126   1.481   0.1385
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234  on 80  degrees of freedom
Residual deviance: 54.402  on 73  degrees of freedom
AIC: 70.402

Number of Fisher Scoring iterations: 6
```

As we can see, the interaction terms of the model do not seem to indicate interaction. Among the rest of the predictors, only “Age” seems to be significant. Now, let's test more specifically the interactions with an analysis of deviance:

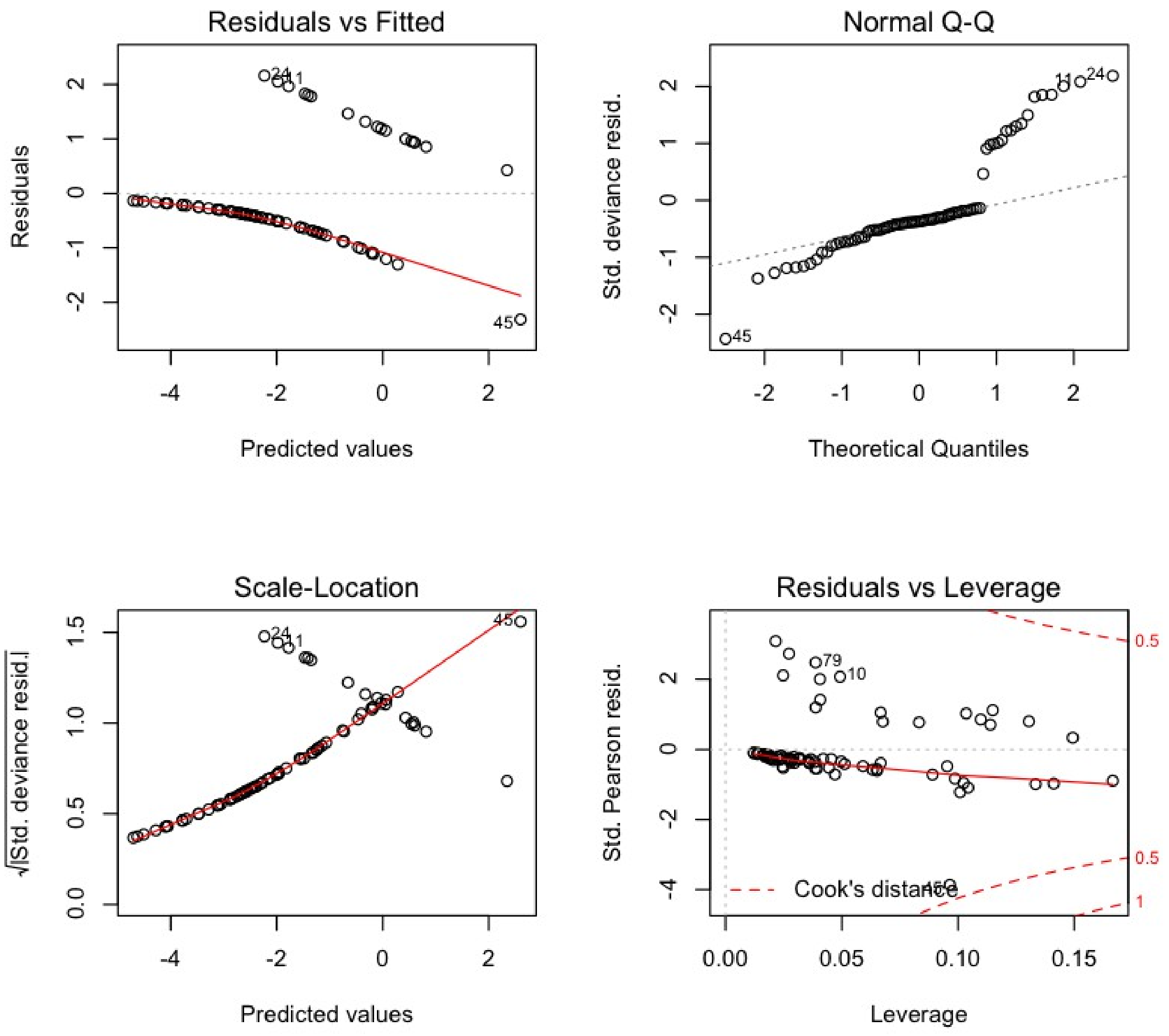
```
> anova(mod1,test="Chisq")
Analysis of Deviance Table

Model: binomial, link: logit

Response: Kyphosis

Terms added sequentially (first to last)


                 Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                               80     83.234
Age               1   1.3020       79     81.932   0.25385
Start             1  16.6334       78     65.299 4.534e-05 ***
Number            1   3.9191       77     61.380   0.04774 *
Age:Start         1   0.9370       76     60.443   0.33305
Age:Number        1   3.4418       75     57.001   0.06357 .
Start:Number      1   0.0069       74     56.994   0.93400
Age:Start:Number  1   2.5922       73     54.402   0.10739
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

This test is adding sequentially to a null model (only with intercept) each of the predictors. After adding the predictors “Age”, “Start”, and number, the rest of the terms do not contribute significantly to the reduction of deviance. So, we can definitely forget about the interactions, and we reformulate mod1 without interactions, only additive terms:

```
> mod1<-glm(Kyphosis~Age+Start+Number,family=binomial,data=kyphosis)
> summary(mod1)

Call:
glm(formula = Kyphosis ~ Age + Start + Number, family = binomial,
    data = kyphosis)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3124  -0.5484  -0.3632  -0.1659   2.1613

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.036934   1.449575  -1.405  0.15996
Age          0.010930   0.006446   1.696  0.08996 .
Start       -0.206510   0.067699  -3.050  0.00229 **
Number       0.410601   0.224861   1.826  0.06785 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234  on 80  degrees of freedom
Residual deviance: 61.380  on 77  degrees of freedom
AIC: 69.38

Number of Fisher Scoring iterations: 5
```

“Start” is significant, but the other predictors are not (although they are close to be). The AIC is better than the previous version with parameters. We should now check the model plots:

```
> par(mfrow=c(2,2))
> plot(mod1)
```



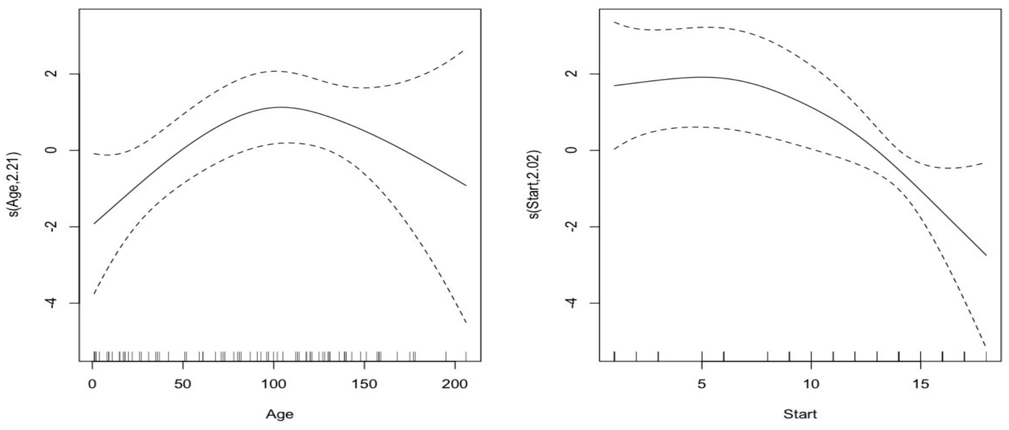
These model plots do not look good at all. Now, we are going to try to fit a series of GAM models. We start with a model with smoothing functions for all our predictors except “Number”. The reason is that this predictor has only one data for two of its levels, and then the smooth function is not going to work on those levels. We should better add it as a linear covariate (in the output, it will be treated then as a parametric predictor):

```
> mod2<-mgcv::gam(Kyphosis~s(Age)+s(Start)+Number,family=binomial,data=kyphosis)
> summary(mod2)

Family: binomial
Link function: logit

Formula:
Kyphosis ~ s(Age) + s(Start) + Number

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -3.5926     1.1464  -3.134  0.00172 **
Number        0.3333     0.2324   1.434  0.15148
---
```

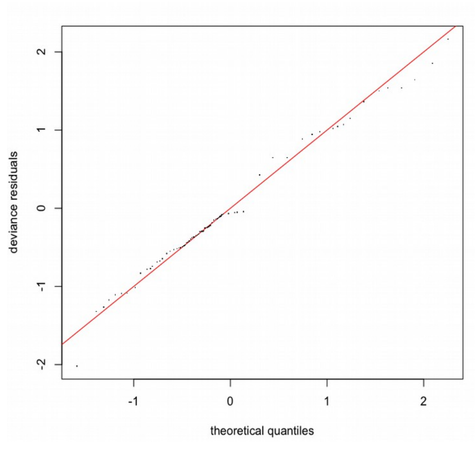
```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
           edf Ref.df Chi.sq p-value
s(Age)   2.209  2.790  6.305  0.0838 .
s(Start) 2.020  2.523  9.645  0.0149 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.355   Deviance explained = 39.4%
UBRE score = -0.22384  Scale est. = 1         n = 81
```

It seems that “Number” is not significant, as well as “Age” (although this is close to significance). We should plot the model now in order to check for these two variables were smoothed:

```
> par(mfrow=c(1,2))
> plot(mod2)
```



So the smoothed “Age” is close to a flat line, but still has some sort of clear “maximum”, that is why it s close o be significant (a flat line would be not significant). On the other hand, “Start” shows a clear trend that is well fitted with this spline. We can have a look at some additional plots:



This is way better than our previous QQ plot from glm. Since the error distribution is not normal, some deviations from it are expected, but not as much as in mod1.

Now, we should try new models without those predictors that do might not be useful ("Age", "Number") and compare all our models in order to select the best one. This is very important, because in GAM models, the p – values are only approximations, so the best way to test the influence of predictors is test the performance of models with and without these predictors.

```
> mod3<-mgcv::gam(Kyphosis~s(Age)+s(Start),family=binomial,data=kyphosis)
> summary(mod3)

Family: binomial
Link function: logit

Formula:
Kyphosis ~ s(Age) + s(Start)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -2.1723     0.4846  -4.483 7.37e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
           edf Ref.df Chi.sq p-value
s(Age)   2.183  2.761  6.348  0.0805 .
s(Start) 2.199  2.740 12.905  0.0041 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =   0.33   Deviance explained = 37.1%
UBRE score = -0.2204  Scale est. = 1         n = 81
```

The R squared and deviance explained are both a bit lower, but model does not seem to change a lot. Now, a model with only "Start":

```
> mod4<-mgcv::gam(Kyphosis~s(Start),family=binomial,data=kyphosis)
> summary(mod4)

Family: binomial
Link function: logit

Formula:
Kyphosis ~ s(Start)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.8951     0.4304  -4.403 1.07e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
           edf Ref.df Chi.sq p-value
s(Start) 2.209  2.756  13.22  0.0036 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

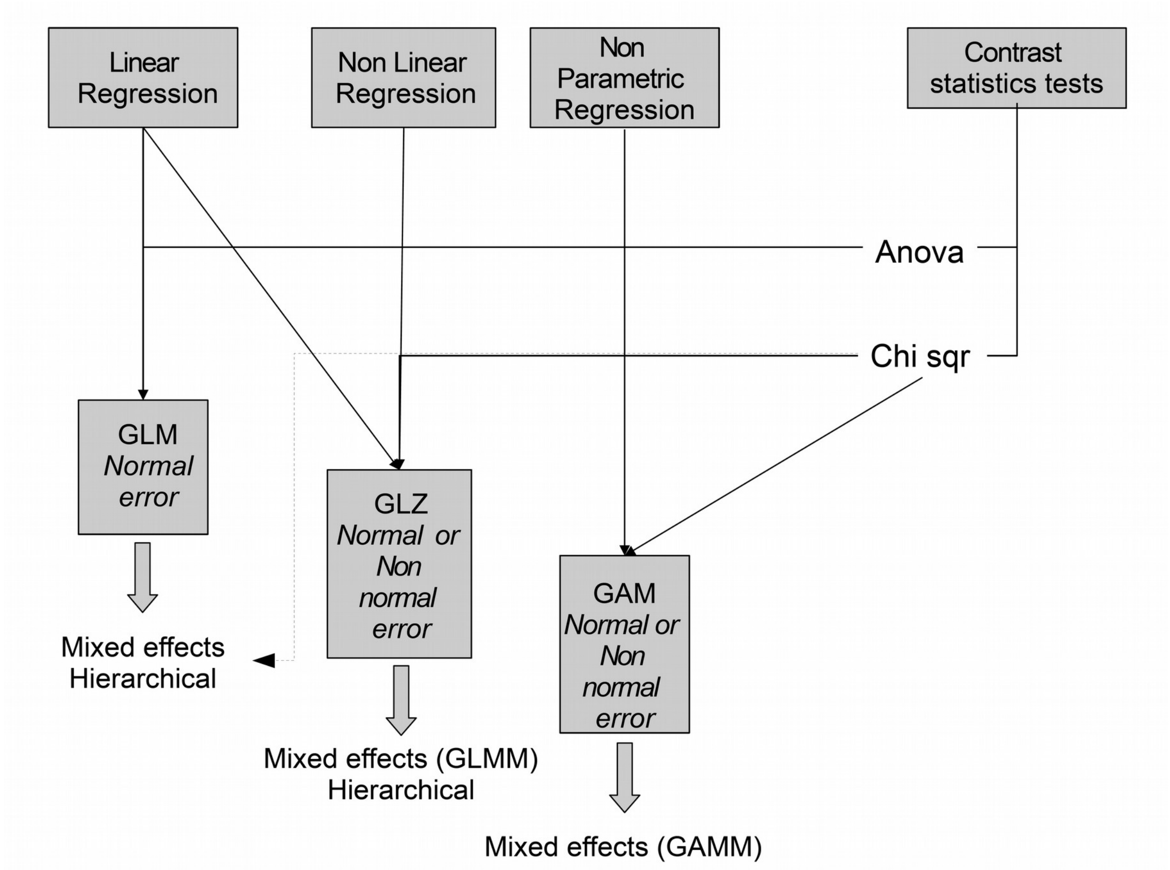
```
R-sq.(adj) =  0.222   Deviance explained = 25.6%
UBRE score = -0.15665  Scale est. = 1         n = 81
```

In this case, the model seems to perform significantly worse. Let's test it with a deviance test for all these 3 GAM models (the GLZ is obviously outperformed by these 3):

```
> anova.gam(mod2,mod3,mod4,test="Chisq")
Analysis of Deviance Table

Model 1: Kyphosis ~ s(Age) + s(Start) + Number
Model 2: Kyphosis ~ s(Age) + s(Start)
Model 3: Kyphosis ~ s(Start)
  Resid. Df Resid. Dev       Df Deviance Pr(>Chi)
1    74.770     50.410
2    75.618     52.383 -0.84749  -1.9735  0.13059
3    77.791     61.893 -2.17293  -9.5094  0.01046 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

There is no significant difference between mod2 and mod3 (named Model 1 and Model 2 in the output, respectively), but there is indeed between mod4 and the others. So, accordingly with the principle of maximum parsimony, we have to select mod3, with only two predictors, both smoothed with splines.

So, with GAM models (that may include GAMM as well) we are done with our overview of the main frames for statistical modeling. We can add the GAM component to our general scheme:


Linear Regression
Non Linear Regression
Non Parametric Regression
Contrast statistics tests
Anova
Chi sqr
GLM *Normal error*
GLZ *Normal or Non normal error*
GAM *Normal or Non normal error*
Mixed effects Hierarchical
Mixed effects (GLMM) Hierarchical
Mixed effects (GAMM)

4.3 Introduction to Meta – Analysis

The techniques of Meta – Analysis are aimed to synthesize **quantitatively** research performed in different experiments, by different researchers, around a same hypothesis. These techniques treat each experiment as a single data for our analysis, synthesizing their results with a few measures. Then they perform a single analysis in order to test the hypothesis with the, in principle, increased power of including all these results. One of the main problems is gathering all the data needed for this analysis, which consist mainly in an exhaustive work of literature review. Then, a synthesizing measure has to be established in order to standardize the results from different experiments. When we have all these data ready, we can perform the test. The test, in principle, could be performed within any framework of statistical modeling among those that we have seen, from simple linear regression to GAM. But the most common are simple linear models of the kind of GLM. Meta – analysis is showing increased interest among scientists from different disciplines, since it is a good tool for synthesizing knowledge from large amounts of data sets and experiments.
Meta – analysis improve previous methods for quantitative reviews, which were very simple, like “vote counting” or “combined probabilities” methods.
It is not a technique that comes to substitute or improve qualitative reviews, rather, complements them. Meta – analysis usually do not produce much conceptual avance, but they have certain features that complement qualitative reviews: it gives quantitative measures, it is repeatable and can balance the studies included by weighting experiments in function of their sample size. But, it has also the disadvantage of being influenced by the studies included in the analysis. This is why the literature review task is very important.
The general steps to perform a meta – analysis are the following:

- Literature search: This a fundamental step, you have to reach as many studies as possible in order to avoid publication bias. The problem is that even with today's data bases, there are many studies that are difficult to reach, such as reports, master or Ph. D. dissertations... Another aspect to consider is whether to reject including certain studies based in their quality (critics say “garbage in – garbage out”). The problem is that criterions for rejecting studies have to be objective (for instance, reject experiments without controls or with pseudo – replication).

- Measures of effect size: the “effect size” is a standardized measure of the effect on the response variable of interest of the predictors that we are considering. We have to extract the value of effect size for all our studies. Several common measures of effect size are: the odds ratio (ratio of the probability that the event occurs/does not occur), correlation coefficient, standardized mean difference (there are several formulas, some are similar to the *t* statistic), Log - response ratio (LN(average treatment/average control). Other measures of effect size could be p – values, or certain statistics (*t* statistic, *F* statistic) which in fact are standardized measures of effect size. But these latter are less preferred. Then, a database has to be constructed with all the references, information about experimental design of each work, factors, sample sizes, etc. and the corresponding measures of effect size.

- Weighting effect sizes: this could be performed or not, but it is recommended when we have large differences in sample size among studies.

- Calculate averages and confidence intervals for effect sizes and contrast hypothesis.

Now, we are going to perform these tasks (except the literature review and database construction) with an example data set from R.

Example **(file R.S4.3)**

Available R packages to perform meta – analysis are *meta*, *rmeta* and *metafor*. We are going to use *metafor*. This package provides methods to conduct meta – analysis with different model designs: contingency 2x2 tables, regression, GLM and GLZ, both with random effects. It has many data sets that could be used as an example. Here, we will use the data set *dat.curtis1988* which contains a data base with experiments testing the effect of elevated CO_2 levels on woody plant mass.
First thing is to load the package (having a look at the help for a package is also a good idea when we don't know much about package functions).

```
> require(metafor)
> ?metafor
> data(dat.curtis1998)
> head(dat.curtis1998)
  id papnum   genus     species function co2.ambi co2.elev units time pot method
1 21     44   ALNUS       RUBRA    N2FIX      350      650  ul/l   47 0.5     GC
2 22     44   ALNUS       RUBRA    N2FIX      350      650  ul/l   47 0.5     GC
3 27    121    ACER      RUBRUM    ANGIO      350      700   ppm   59 2.6     GH
4 32    121 QUERCUS     PRINUS    ANGIO      350      700   ppm   70 2.6     GH
5 35    121   MALUS   DOMESTICA    ANGIO      350      700   ppm   64 2.6     GH
6 38    121    ACER SACCHARINUM    ANGIO      350      700   ppm   50 2.6     GH
  stock xtrt   level      m1i       sd1i n1i    m2i       sd2i n2i
1  SEED FERT    HIGH  6.8169 1.7699820   3 3.9450 1.1157970   5
2  SEED FERT CONTROL  2.5961 0.6674662   5 2.2512 0.3275839   5
3  SEED NONE       .  2.9900 0.8560000   5 1.9300 0.5520000   5
4  SEED NONE       .  5.9100 1.7420000   5 6.6200 1.6310000   5
5  SEED NONE       .  4.6100 1.4070000   4 4.1000 1.2570000   4
6  SEED NONE       . 10.7800 1.1630000   5 6.4200 2.0260000   3
```

So this data set contains the following information (it can be consulted in the help for this data set): *id* is the observation number, *papnum* is the paper identification (several observations might come from the same paper), *genus* and *species* are taxonomic information about the involved species, *function* are the functional groups that the plants belong to. *co2.ambi* and *co2.elev* are the CO_2 levels of the control and treatment, respectively. *units* are the units of CO_2, *time* is the time of exposure to CO_2, *pot* is the growing method, *method* is the CO_2 exposure facility, *stock* is the planting stock code, *xtrt* is the interacting treatment code, *level* is the interacting treatment level codes, *m1i* is mean plant mass under elevated CO_2 level (the treatment), *sd1i* is the standard deviation of plant mass under elevated CO_2 level, *n1i* is the number of observations under elevated CO_2 level, *m2i* is the mean plant mass under ambient CO_2 level (the control), *sd2i* the standard deviation of plant mass under ambient CO_2 level and *n2i* the number of observations under ambient CO_2 level.
Now it is time to choose an appropriate measure of effect size. The function that we need is escalc(). We need to look at the help from this function in order to know how to select an

appropriate measure of effect size.

```
> ?escalc
```

The key argument is “measure”, the actual measure of effect size to use. Looking at the “details” section of the help page, we can see which kinds of measures of effect size are available. We need a measure for quantitative variables. We could use the mean difference, standardized, because the CO_2 units are not the same, and, mainly, the CO_2 level of the controls are not the same. There is the choice as well for the mean differences with heteroscedastic groups (different group variances). But we are going to choose the log transformed ratio of means, which is a very common measure of effect size. Many statisticians like this measure because it is centered around 0 (when the 2 means have the same value) with a distribution that is close to Normal. Because the experiments have different sample sizes, we should better weight the effect sizes with these sample sizes. The code for this function becomes then:

```
> ES<-
escalc(measure="ROM",n1i=n1i,n2i=n2i,m1i=m1i,m2i=m2i,sd1i=sd1i,sd2i=sd2i,data=dat.curtis1
998)
```

The actual effect sizes and its sample variances are returned as the two last columns. Now, we are going to fit a GLM model with these effect sizes. The function that we need is rma(). In the help pages and manual of this package, the predictor variables are called “moderators”.
As you could imagine, in our data there are many variables that could be used as fixed factors (genus, species, functional groups, CO_2 exposure facility, the interacting treatment) covariates (time of exposure). Random factors (genus, species or functional groups could be random factors, depending on the researcher point of view or knowledge about the data, growing method). The correct procedure would be to fit several models and compare them with, for instance AIC and log – likelihood ratio test (there is the function anova.rma.uni() in this package). But because we are only introducing meta – analysis in this course, we are simply going to fit a couple of simple reasonable models and use them as an example:

```
> mod1<-rma(yi=ES[,21],vi=ES[,22],mods=~function.+time+method,data=ES,method="FE")
> summary(mod1)

Fixed-Effects with Moderators Model (k = 102)

   logLik   deviance        AIC        BIC       AICc
-190.6529   658.4083   393.3058   409.0556   394.1900

Test for Residual Heterogeneity:
QE(df = 96) = 658.4083, p-val < .0001

Test of Moderators (coefficient(s) 2,3,4,5,6):
QM(df = 5) = 110.6102, p-val < .0001

Model Results:

                             se     zval     pval    ci.lb    ci.ub
```

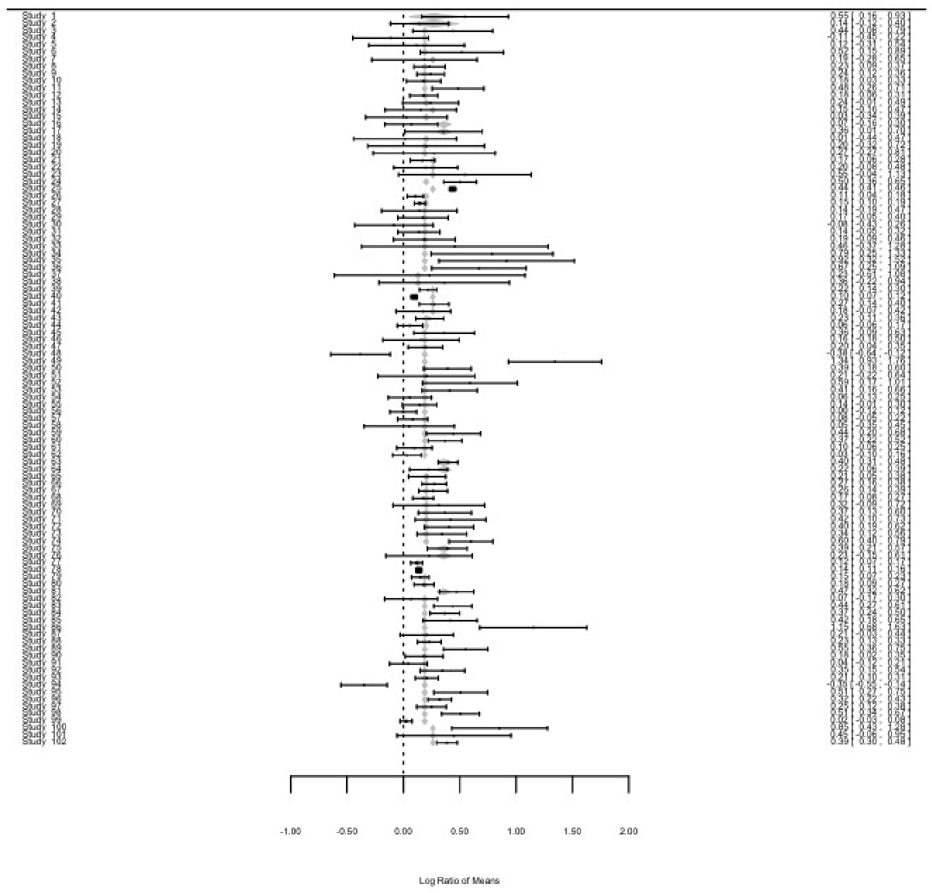
```
intrcpt          0.2665  0.0089  29.9665  <.0001   0.2490   0.2839  ***
function.GYMNO  -0.0647  0.0120  -5.3977  <.0001  -0.0882  -0.0412  ***
function.N2FIX   0.0044  0.1099   0.0397  0.9684  -0.2111   0.2198
time            -0.0000  0.0000  -1.8553  0.0636  -0.0001   0.0000    .
methodGH        -0.0648  0.0119  -5.4326  <.0001  -0.0882  -0.0414  ***
methodOTC        0.1025  0.0315   3.2486  0.0012   0.0406   0.1643   **

---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

In the help page of rma() we can check the various options to code the model. We used all these predictors (“moderators”) as fixed effects, so we must set the argument “method” to “FE”.
In the model results, the first value are the coefficients for each predictor. In the case of the intercept, its value corresponds to the average “basal” size effect of our analysis. The other coefficients show how each predictor increases or reduces its value. The interpretation is the same as what we saw for linear regression and GLM.
The test for the residual heterogeneity suggest that residuals are not homogeneous. The test for the moderators (predictors) suggest that they are overall significant, but looking at the coefficients, it seems that the moderator “time” may not be significant. So, it is not a bad idea to code a new model without “time” and compare it to the previous one with the log – likelihood ratio test:

```
> mod2<-rma(yi=ES[,21],vi=ES[,22],mods=~function.+method,data=ES,method="FE")
> summary(mod2)

Fixed-Effects with Moderators Model (k = 102)

   logLik   deviance        AIC        BIC       AICc
-192.3739   661.8503   394.7478   407.8727   395.3728

Test for Residual Heterogeneity:
QE(df = 97) = 661.8503, p-val < .0001

Test of Moderators (coefficient(s) 2,3,4,5):
QM(df = 4) = 107.1682, p-val < .0001

Model Results:

                             se     zval    pval    ci.lb    ci.ub
intrcpt          0.2616  0.0085  30.7903  <.0001   0.2449   0.2783  ***
function.GYMNO  -0.0593  0.0116  -5.1007  <.0001  -0.0821  -0.0365  ***
function.N2FIX   0.0072  0.1099   0.0652  0.9480  -0.2082   0.2226
methodGH        -0.0720  0.0113  -6.3811  <.0001  -0.0941  -0.0499  ***
methodOTC        0.0976  0.0314   3.1064  0.0019   0.0360   0.1592   **

---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> anova(mod1,mod2,test="Chisq")
        df      AIC      BIC     AICc    logLik    LRT   pval       QE
Full     6 393.3058 409.0556 394.1900 -190.6529                658.4083
Reduced  5 394.7478 407.8727 395.3728 -192.3739 3.4420 0.0636 661.8503
```

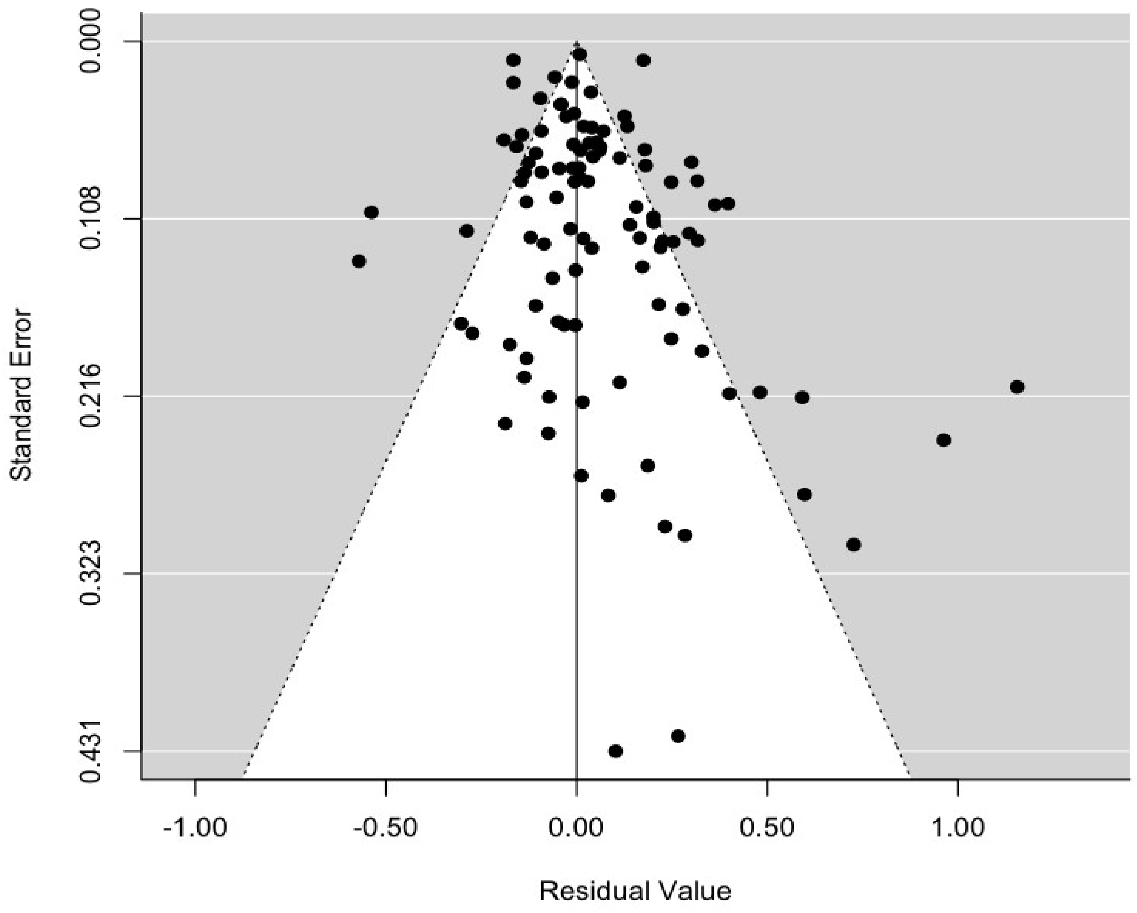
The first model has slightly lower AIC value, but this difference is not significant, so we should better choose mod2, that has one predictor less.
An important plot to evaluate a Meta – Analysis is the "forest plot", that represents all the data of size effects with their corresponding 95% confidence intervals.

```
> forest(mod2)
```



In this plot, each experiment (data point) has an area proportional to its sample size (that we used to weight data). The dotted line follows the 0 value of log ratio of means (0 means no effect). As you can see, most of our average effect sizes are above this value of 0, but not all the confidence intervals. So, in most cases there is a significant positive effect, and all our fixed factors have a significant effect, as we saw earlier. We can conclude that there is a positive significant effect of CO_2 levels on woody mass, that is modulated by several predictors in different senses. But we should have a look as well at other techniques of model diagnosis. The funnel plot is a common way to represent Meta – Analysis data that tells us two things: First, check about our potential "publication bias". "publication bias" consist in the bias to publish works that either demonstrate or refute our hypothesis under study. In principle, it is expected that small scale studies are more prone to publication bias, whereas lasrge scale studies are always published, irrespective of the final conclussion. Second, the presence of heterogeneity. The presence of heterogeneity was already suggested by the previous Q test, but here we can check it visually, together with publication bias:

```
> funnel(mod2)
```



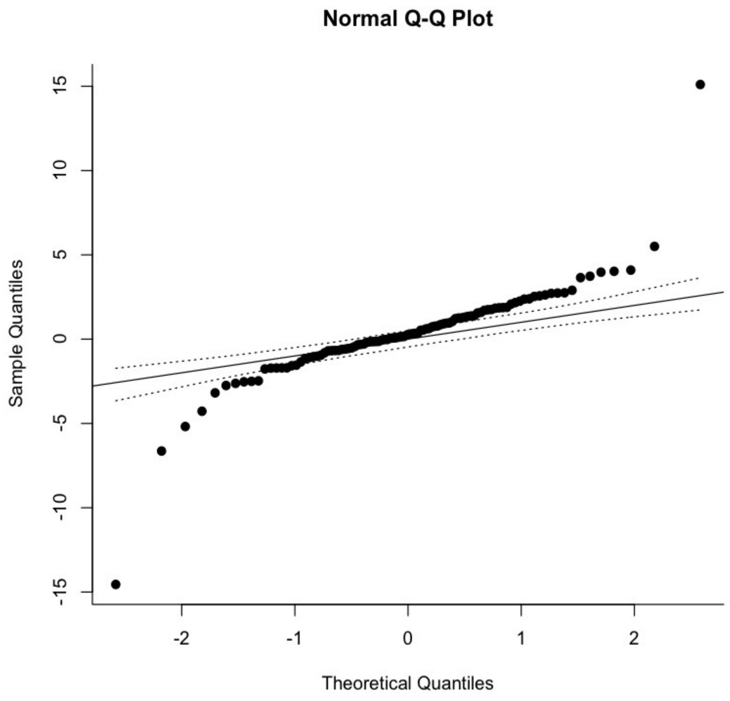
In this plot we can see the “funnel” created by pseudo – confidence intervals for the model residuals and their standard error. The lower the absolute value of the residuals, the lower should be the standard error. So if residuals are heterogeneous in their errors, they should increase their error in a constant ratio as they increase their absolute value (greater residuals have greater error). This is what creates a “funnel” shape symmetric respect to the 0 value for residuals. There are other magnitudes that could be represented as well in this plot, and show a “funnel” shape as well. This “funnel” shape would be distorted if two things happened:

- If residuals are not homogeneous, and their errors do not increase with their absolute value.
- If there was some sot of publication bias towards the publication of either positive or negative results. If there is no bias, both “sides” of the funnel should look the same (we may then have an asymmetric funnel).

In this case, we might have both problems, residual heterogeneity (already suggested by the Q test, as we mentioned earlier) and bias towards the exclusion of those studies with

smaller sample size (higher standard error). In this particular case, those studies that show no effects of increasing CO_2.
A QQ plot is also another good diagnosis plot for our model:

```
> qqnorm(mod2)
```



The plot shows problems with normality. The problems with residual heterogeneity and normality could be solved if we are able to find a better model with new predictors, perhaps some random effects, etc. but the problem with publication bias is not solvable.

Bibliography

- *A Primer of ecological Statistics.* N.J. Gotelli, A.M. Ellison. Second edition. Sinauer Associates Inc. 2013.

- *Experimental Design and Data Analysis for Biologists.* G.P. Quinn, M.J. Keough. Cambridge University Press, 2002.

- *Introductory statistics with R*. Peter Dalgaard. Springer. 2002.

- *A beguinners guide to R (use R!)*. Alain Zuur, Elena N. Ieno, Erik H.W.G. Meesters. Springer. 2009.

- Non linear regression with R. C. Ritz, J.C. Streibig. Springer 2008.

- *Mixed Effects Models and Extensions in Ecology with R.* A. Zuur, E.N. Ieno, N. Walker, A.A. Saveliev, G.M. Smith. Springer, 2009.

- Multiple Comparisons Using R. F. Bretz, T. Hothorn, P. Westfall. Chapman & Hall, 2011.

- *Extending the linear Model with R.* J.J. Faraway. Chapman & Hall, 2006.

- *Generalized Additive Models: An Introduction with R.* S.N. Wood. Chapman & Hall, 2006.

- *Introduction to Meta-Analysis.* M. Borenstein, L.V. Hedges, J.P.T. Higgins, H. Rothstein. Wiley, 2009.

- Applied Meta-Analysis with R. D.G. Cheng, K.E. Peace. Chapman & Hall, 2013.

- *An Introduction to applied Multivariate Analysis with R.* B. Everitt, T. Hothorn. Springer, 2011.

- Numerical Ecology with R. D. Borcard, F. Gillet, P. Legendre. Springer, 2011.

- A Primer of Ecology with R. M. Henry H. Stevens. 2009, Springer.