For-R-casting state-space using R: a simple/practical approach

Giacomo SBRANA

July 6, 2020

Before we start...

R is one of the most powerful (open-source) statistical software since it allows you to do wonderful data analysis. These notes will help you getting familiar with the R language and its potentialities for data analysis. Before starting, it is crucial to install and get familiar with R and R studio. The latter is the one that we will use a lot throughout our classes until the final exam. Please note that R studio does not run without R, therefore R is necessary to work with R studio. Below I report the main steps to install R and then R studio for both Mac and Windows users. A_t

- Mac Users
  - To Install R
  1. Go to www.r-project.org.
  2. Click on "download R" link in the middle of the page under "Getting Started."
  3. Select a CRAN location (no matter which) and click the corresponding link.
  4. Click on the "Download R for (Mac) OS X" link at the top of the page.
  5. Click on the file containing the latest version of R under "Files."
  6. Save the .pkg file, double-click it to open, and follow the installation instructions.
  - To install RStudio.
  7. Go to www.rstudio.com and click on the "Download RStudio" button.
  8. Click on "Download RStudio Desktop."
  9. Click on the last version, recommended for your system, save the .dmg file, open it and then drag and drop it to your applications folder.
- Windows Users

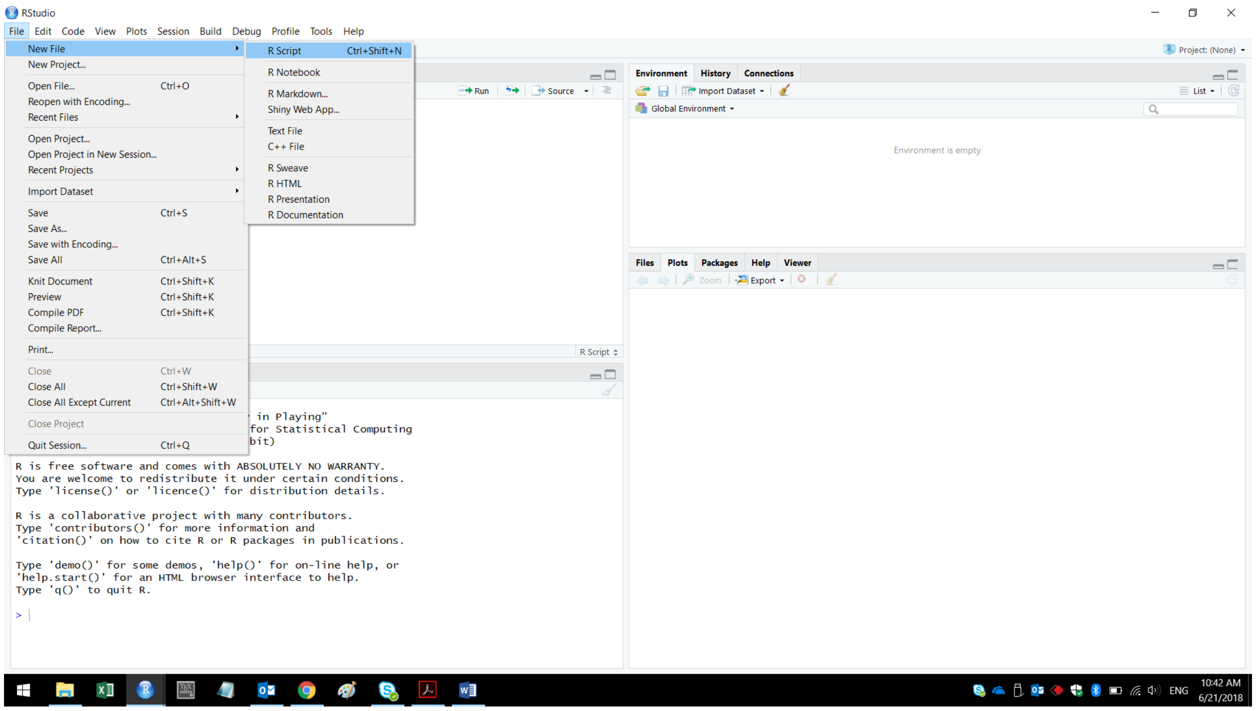
```
+ To Install R:

  1. Go to www.r-project.org.
  2. Click the "download R" link
  3. Select a CRAN location (no matter which) and click the corresponding link.
  4.   Click on the "Download R for Windows" link at the top of the page.
  5. Click on the "install R for the first time" link at the top of the page.
  6. Click "Download R for Windows" and save the executable file and run it (follow the installati
```

  - To install RStudio.
    7. Go to www.rstudio.com and click on "Download RStudio".
    8. Click on "Download RStudio Desktop."
    9. Click on the last version recommended for your system. Then save and run the executable file.

Getting start!

This is what Rstudio looks like when you open it:



```
in Playing"
for Statistical Computing
bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

The top left window allows us to write program script. These scripts can be saved and be reused later.

The Console is the place where we can make quick calculations and where we can visualize our results. When you want to type something for you (or other people) BUT you do not want R to evaluate it you can digit # followed by your comment. Lets practice this by clicking on the console the following:

```
#Hello! Note that below I am running a nice operation. Have a look!

4*6/2
```

```
## [1] 12
```

```
#The result of the operation is the number you see above.
```

Now suppose you want to write a message and then you want to print it. Again, you can install yourself on the console and then you need to run the following:

```
#Below create a message and then I print it

mymessage<-"Tutti frutti!"

print(mymessage)
```

```
## [1] "Tutti frutti!"
```

```
#The message is in Italian
```

RMarkdown

Since we will use a lot RMarkdown, allowing us to make reproducible research and results, please run the following commands in your Console:

```
#To install RMarkdown run the following commands in the Console:

        #install.packages('rmarkdown')
        #install.packages("tinytex")
        #tinytex::install_tinytex()
```

Then check that you can open a simple RMarkdown file following this path:

File –> New File –> RMarkdown File . . .

Then you have a sample template that you can modify. If you then type on "Knit" you are then able to compile your file in HTML or PDF formats.

The Rmarkdown Cheat-sheet is available by clicking on the following website:

https://rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

Below I show you a simple example that create a Rmd file, and further below the output.

You can copy and past the chunk and reproduce the same in your onw Rmd file. Bye the way, if you want to take away the symbol # you need to digit $Ctrl + Shift + c$

```
# ---
#   title: "Simple Rmd example"
# author: "Put your name here"
# date: "May 21, 2020"
# output:
#   pdf_document: default
# html_document: default
# ---
#
#   ```{r setup, include=FALSE}
# knitr::opts_chunk$set(echo = TRUE)
# ```
#
# #This is a nice example
#
# Hi! This is a picture of me :)
#
# ```{r, out.width = "100px"}
# knitr::include_graphics("Giaco.jpg")
# ```
# This is an example to show how you can write beuatiful documents in Rmarkdown.
# Suppose we have a variable of interest $y_t$ and we believe that it is function
#of $x_t$ as in the following simple linear regression model:
#
#   $$
#   y_t=\tau+\beta \times x_t+\epsilon_t
```

```
# $$
#
#   ```{r,fig.height = 3, fig.width = 4}
# n<-100;eps<-rnorm(n);x<-seq(1,4,length.out =n);y<-3+.8*x+eps
# par(mfrow=c(1,1),bg="yellow")
# plot(y,pch = 21, bg="red", main="This shows my linear regression model")
# lines(3+.8*x)
# ```
#
# Do you know that 3+5=8? Well we can check this using R here: 3+ 5 = ...`r 3+5`

knitr::include_graphics("RmdExample.pdf")
```

Simple Rmd example

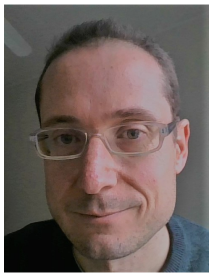
Put your name here

May 21, 2020

This is a nice example

Hi! This is a picture of me :)

```
knitr::include_graphics("Giaco.jpg")
```



This is an example to show how you can write beuatiful documents in Rmarkdown. Suppose we have a variable of interest y_t and we believe that it is function of x_t as in the following simple linear regression model:

$$y_t = \tau + \beta \times x_t + \epsilon_t$$

```
n<-100;eps<-rnorm(n);x<-seq(1,4,length.out =n);y<-3+.8*x+eps
par(mfrow=c(1,1),bg="yellow")
plot(y,pch = 21, bg="red", main="This shows my linear regression model")
lines(3+.8*x)
```

1

Working directory

It is important to establish the working directory as it will be the space where you will save data, scripts etc. . . For example, suppose you want to work with your memory stick (USB) this is the command to digit:

```
#The command below establish the working directory
#setwd("C:/Users/giacomo.sbrana/blabla")
# The command below is useful since it remainds you...
#which is the working directory you are :)
getwd()
```

```
## [1] "C:/Users/giacomo.sbrana/OneDrive - NEOMA Business School/Documents/Liang/AppliedTimeSeries/ForS
```

```
#Note that when you insert the working directory we do not use \ but / to recall subfolders
```

You can use R studio for many types of analysis. This is an easy example that show how to run some basic calculations and assign value to a specific object.

```
# Below I create object that contain a scalar information.
a<-4
b<-25
pro=a*b
div=pro/2
```

In the above expression I use "<-" and "=" indiscriminately as equal. Please note that these two symbols make the same job: they assign to an object on the left-hand side the object on the right hand side.

Save your Script

If you want to save your code, in order to use it later, go to "File –> Save as" and give it the name you want. Just remember that the extension of the Script is ".R" For example suppose you save a file and you call it MYFILE, then your script is saved in the working directory as follows: "MYFILE.R"

Save your data

Suppose that you want to save the dataset your are working on. Note that the data appear in the top-right-hand window called "Environment".

If you click on the disk icon (to save the data), suppose you want to call your dataset MYDATA, your data is saved in the working directory as "MYDATA.Rdata". You can then reopen your dataset by clicking on the first icon open below the Environment.

Help when you do not know a command

Suppose you want to know what a command do when you use it. For example you want to know what the command "sum()". You can do it for in two ways:

?sum

help(sum) Internet is a great source for learning how to use R! One of the asset of R is that you find everything on internet. If you do not know how to run something, you go to internet and you will find what you need in 99% of the cases (the remaining 1% is concerned with those that do very complicated stuff).

Vectors, Matrix and other stuff

When you digit $a < -3$ in the console you create an object containing one element only. Suppose you want to create an object containing more than one element. For example, suppose you want to create a vector containing a column (or row) of elements. Imagine this vector $v = [2, 6, 1, 3, 11]$. You can create this by clicking:

```
##The command below create a vector (a column of numbers)
v<-c(2,6,1,3,11)
v
```

```
## [1]  2  6  1  3 11
```

```
# If you want to create a row containing the same numbers
# you can transpose the vector v using the command t().
myrow<-t(v)
myrow
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    6    1    3   11
```

Note that the command t() allows transposing a vector or a matrix (this will be discussed below).The vector v is identified using [1] followed by the numbers.

Another way to create a vector is create a sequence of number (in a column) as follows:

```
##The command below create a vector (a column of numbers)
seq(from=-4, to=5,by=1.5)
```

```
## [1] -4.0 -2.5 -1.0  0.5  2.0  3.5  5.0
```

```
## if you do not specify by= it is assumed by 1. as shown here
seq(from=-4, to=5)
```

```
##  [1] -4 -3 -2 -1  0  1  2  3  4  5
```

```
## If you want a sequence with a specific length than you have to do:
seq(4, 5, length.out = 5)
```

```
## [1] 4.00 4.25 4.50 4.75 5.00
```

Another way to create a vector is create a repetition of numbers (in a column) as follows:

```
##The command below create a vector (a column of numbers) repeting 5 times the number 2.
rep(2,5)
```

```
## [1] 2 2 2 2 2
## This commands repets 4 times the number 3.
rep(3,4)
```

```
## [1] 3 3 3 3
```

Finally, a vector can be considered as a column of a row of a matrix. The latter can be created in R as follows:

```
##The command below create a 3 by 3 matrix repeating 4 across each element of the matrix.
matrix(4,3,3)
```

```
##      [,1] [,2] [,3]
## [1,]    4    4    4
## [2,]    4    4    4
## [3,]    4    4    4
```

```
##The command below create a 2 by 2 matrix with the elements of a vector.
matrix(c(11,5,9,2),2,2)
```

```
##      [,1] [,2]
## [1,]   11    9
## [2,]    5    2
```

```
##The command below create a vector  matrix with the elements of a vector.
matrix(c(1,2,3,4),4,1)
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
```

```
##The command below create a vector  matrix with the elements of a vector.
matrix(c(1,2,3,4),1,4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
```

You probably noted that, for example [3,], identifies the third row while [,2] identifies the second column.

More generally, don't forget the following commands that are needed to extract data from a matrix or a vector.

x[n]: the nth element of a vector

x[m:n]: the mth to nth element

x[c(k,m,n)]: specific elements

x[x>m & x<n]: elements between m and n

x[["n"]]: idem

[i,j]: element at ith row and jth column

[i,]: row i in a matrix

x[-c(3,5)]: x excluding the third and fifth elements

```
v1<-c(21,5,2,15)
v2<-c(-3,-6,1,-7)
v3<-c(102,10,-13,4)
x<-matrix(cbind(v1,v2,v3),4,3)
x
```

```
##      [,1] [,2] [,3]
## [1,]   21   -3  102
## [2,]    5   -6   10
## [3,]    2    1  -13
## [4,]   15   -7    4
```

```
x[2] #the nth element of a vector
```

```
## [1] 5
```

```
x[4:7] ### the mth to nth element
```

```
## [1] 15 -3 -6  1
```

```
x[c(2,9,4)]###  specific elements
```

```
## [1]   5 102  15
```

```
x[x>0&x<12]### : elements between m and n
```

```
## [1]  5  2  1 10  4
```

```
x[[11]]### : idem
```

```
## [1] -13
```

```
x[3,2] ### : element at ith row and jth column
```

```
## [1] 1
```

```
x[2,]### : row i in a matrix
```

```
## [1]  5 -6 10
```

```
x[3:4,1:2] ### : submatrix
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]   15   -7
```

```
v3[-c(1,2,4)] ##excluding elements
```

```
## [1] -13
```

Create factors

Sometimes you want to create a factor. For example, suppose that you run a questionnaire and that you need to register if the person is a female or a male. You can create a factor using this:

```
whichsex<-c("Female","Male","Male","Female","Male")
sex<-factor(whichsex,levels = c("Female","Male"))
sex
```

```
## [1] Female Male   Male   Female Male
## Levels: Female Male
```

Suppose you registered the sex using numbers such that Female is 0 and Male is 1.

```
thesex <- c(round(runif(7,0,1)))
thesex
```

```
## [1] 0 1 1 1 1 0 1
```

```
sex<- factor(thesex,labels=c("Female","Male"))
sex
```

```
## [1] Female Male   Male   Male   Male   Female Male
## Levels: Female Male
```

If your variable can be ordered in terms of importance you can use the command *ordered* as follows:

```
set.seed(223)
edu <- c(round(runif(10,1,3)))
education<- ordered(edu,labels=c("Undergrad","Postgrad","Phd"))
education
```

```
##  [1] Phd       Postgrad  Undergrad Postgrad  Postgrad  Phd       Postgrad
##  [8] Postgrad  Phd       Phd
## Levels: Undergrad < Postgrad < Phd
```

Removing elements of a vector/matrix

Suppose you have a vector and you want to remove some elements:

```
myvector<-c(4,22,56,77,26,88,100)
myvector
```

```
## [1]   4  22  56  77  26  88 100
```

```
myvector[-4]
```

```
## [1]   4  22  56  26  88 100
```

```
myvector[-c(1,3)]
```

```
## [1]  22  77  26  88 100
```

```
set.seed(124)
M<-matrix(rnorm(16),4,4)
M
```

```
##             [,1]       [,2]       [,3]       [,4]
## [1,] -1.38507062  1.4255380  0.1970939 -0.4050909
## [2,]  0.03832318  0.7444798  1.2071538  0.9953866
## [3,] -0.76303016  0.7002294  0.3183367  0.9588178
## [4,]  0.21230614 -0.2293546 -1.4237989  0.9180879
```

```
M[-c(2,4),]
```

```
##            [,1]      [,2]      [,3]       [,4]
## [1,] -1.3850706 1.4255380 0.1970939 -0.4050909
## [2,] -0.7630302 0.7002294 0.3183367  0.9588178
```

```
M[-c(1,4),-c(1,2)]
```

```
##           [,1]      [,2]
## [1,] 1.2071538 0.9953866
## [2,] 0.3183367 0.9588178
```

Sometimes you need to create an array, for example an object containing a series of matrices. You can accomplish this using the command *array*. Example:

```
array(1:3, c(2,4)) # This is like create a matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    2    1
## [2,]    2    1    3    2
```

```r
array(1:8,c(2,2,3)) # This is a object containing three matrices
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
##
## , , 3
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

Interestingly, the function array can be used also to create a matrix

Matrix operations

You can add/subtract/multiply/divide vectors with the same dimensions. Some examples are below:

```r
v1<-c(3,2,6)
v2<-c(-2,-1,5)
v1+v2
```

```
## [1]  1  1 11
```

```r
v1-v2
```

```
## [1] 5 3 1
```

```r
v1*v2
```

```
## [1] -6 -2 30
```

```r
v1/v2
```

```
## [1] -1.5 -2.0  1.2
```

You cannot add/subtract/divide vectors of different dimensions. You can multiply vectors with different dimensions by using the symbol %*%

```
v1<-c(3,2,6)
v2<-c(-2,-1)
v1%*%t(v2)
```

```
##      [,1] [,2]
## [1,]   -6   -3
## [2,]   -4   -2
## [3,]  -12   -6
```

For matrices you can add/subtract/multiply/divide vectors with the same dimensions. You cannot add/subtract/divide matrices of different dimensions. You can multiply matrices with different dimensions such as m1%*%m2 ONLY IF the number of columns of m1 corresponds to the number of rows of m2

```
m1<-matrix(rnorm(6),2,3)
m2<-matrix(rnorm(9),3,3)
m1%*%m2
```

```
##           [,1]       [,2]      [,3]
## [1,] -1.279957  0.7353306 0.9001165
## [2,] -1.554485 -0.1157825 1.7718876
```

For matrices the concept of division is linked to the concept of inverse matrix. The matrix inverse is calculated with this $solve(m)$. You can do m1%*%solve(m2) ONLY IF m2 is a square matrix $no.rows = no.cols$.

```
set.seed(123)
m1<-matrix(rnorm(6),1,2)
m2<-matrix(rnorm(4),2,2)
m1%*%solve(m2)
```

```
##           [,1]      [,2]
## [1,] 0.0385414 0.4570846
```

Here note that m1%*%solve(m2) is something like $m1/m2$ but this is a matrix algebra concept that, for the purpose of this book, we do not need to develop more

Exercise 1:

Create a vector using a sequence going from 21 to 120. Give it the name “a”.

Define b as the length of a.

Define d the 5th element of the vector a.

Define f the vector containing the elements from the 2nd until the 6th.

Define g the vector containing the 1st , 3rd and 7th elements of a.

Create a vector, call it h, containing the sequence of values of "a" from 1 until 100 every 4 observations.

Define i the vector containing the elements bigger than 24 and smaller than 29.

Create a matrix (give it the name l) with 25 rows and 4 columns containing the element of the vector a.

Define m the vector containing the elements of the second column of l.

Define n the vector containing the elements of the third row of l.

Define o the vector containing the elements included from row 6 until row 12 and from column 2 until column 3 of l.

No save all your objects and it to me for the evaluation!

Create a list of objects

Suppose that you have several object of different types and you want to collect them in a list (one object). You can do that using the function *list* as follows:

```
a<-3
b<-c(2,6)
c<-matrix(rnorm(4),2,2)
d<-rep(4,3)
mylist<-list(a,b,c,d)
mylist
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 2 6
##
## [[3]]
##           [,1]      [,2]
## [1,] 1.2240818 0.4007715
## [2,] 0.3598138 0.1106827
##
## [[4]]
## [1] 4 4 4
```

You can now recall the elements of the list as follows: 2, 6 this is the second element.

if you want to remove some elements of a list you can do this:

```
mylist[-c(2,4)]
```

```
## [[1]]
## [1] 3
##
## [[2]]
##           [,1]      [,2]
## [1,] 1.2240818 0.4007715
## [2,] 0.3598138 0.1106827
```

```
#you can also eleminate an element using this:
print("This is another example")
```

```
## [1] "This is another example"
```

```
mylist[[4]]<-NULL
mylist
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 2 6
##
## [[3]]
##           [,1]      [,2]
## [1,] 1.2240818 0.4007715
## [2,] 0.3598138 0.1106827
```

Writing your own function

Suppose you want to create a function you want to use several times or in different context. R allows you to do this. For example:

```
# This is a simple function providing the mean of an object called x.
mymean= function(x){print("Below you find the mean of the variable x"); return(sum(x)/length(x))}
#Example:
x<-c(41,2,14,24,4,22,134)
mymean(x)
```

```
## [1] "Below you find the mean of the variable x"
```

```
## [1] 34.42857
```

```
# This is another function providing the mean of the cross product between two objects (x and y).
crossmean= function(x,y){print("Below you find the mean of the variable x times y");
  return(sum(x*y)/length(x))}
y<-c(-1,32,12,-5,21,-2,0)
crossmean(x,y)
```

```
## [1] "Below you find the mean of the variable x times y"
```

```
## [1] 15.85714
```

Now it is time for you to create new function! We will do this as exercises.

ExerciseCorrelation:

Create three different functions:

One that calculate the variance of a variable.

One that calculate the covariance between two variables.

One that calculate the correlation between two variables.

Hint: Remember that the correlation is the ratio between the covariance of the variables and the product of the standard deviations of them. Also, remember that the covariance of x and y is the mean of x times y minus the product of the mean of x times the mean of y. Also, remember that the standard deviation is the squared root of the variance. Finally, the variance of x is the mean of the squared values of x minus the square of the mean of x. That is Variance of x is equal to $\left[mean(x^2) - (mean(x))^2\right]$. Note that while the correlation function that you created returns the same value of the "cor" function of R, there is a difference between the "var" function and your variance and the "cov" function and your covariance. This is because R divides by n-1 while you divide by n. The n-1 is a correction used for small samples. Here we do not need to discuss this further.

Use the link below if you want to find additional exercises: $http://rstatistics.net/r-lang-practice-exercises-level-1-beginners/$

What have you learnt during this class?

(1) Download R and RStudio.

(2) Open a script, start working on it and save it.

(3) Basic operations/calculations.

(4) How to print a message (involving text).

(5) # is used to write comments. R does not consider all things written on its right hand side.

(6) When you write a path in RStudio do not use instead use /. Example: setwd(“C:/Users/name.surname/D

(7) getwd() #get the current working directory.

(8) setwd() #change the directory to. Example setwd(“c:/RESEARCH/myfiles/”)

(9) The script extension is .R while the data extension is .Rdata

(10) help() #give help regarding a command, e.g. help(hist)

(11) c() #create a vector by concatenating objects, example: x = c(3,5,8,9)

(12) Important: c() create a vector of unknown dimension that can be used for example in a for loop (see later).

(13) 1:19 #create a sequence of integers from 1 to 19

(14) [.] #select elements from a vector or list, e.g. x[2] gives 5, x[c(2,4)] gives 5 9 for x as above

(15) matrix(1:20,4,5) #Create a matrix with 4 rows ans 5 cols.

(16) m[,3] #gives the 3rd column of the matrix m

(17) m[2,] #gives the 2nd row of the matrix m

(18) = or <- #assign something to a variable, example: x=c(1:5) is equivalent to x<-c(1:5)

(19) sum() #get the sum of the values in x by sum(x)

(20) mean() #get the mean of the values in x by mean(x)

(21) sd() #get the standard deviation of the values in x

(22) var() #get the variance of the values in x

(23) cor() #gives the linear correlation coefficient

(24) seq() #create a sequence. Example seq(2,11,by=3) create a sequence from 2 to 11 with increments equal to 3

(25) rep() #repeat n times the value x, e.g. rep(2,5) gives 2 2 2 2 2

(26) How to create your own function. Ex. name= function(x){operation}. This create a function that takes x as input and return the operation desired.

(27) How to create an array

(28) How to create a list

```r
#Ex1
for(i in 1:4){print(i)}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

```r
#Ex2
for(i in c(-3,0,5,11)){print(rep(i,2))}
```

```
## [1] -3 -3
## [1] 0 0
## [1] 5 5
## [1] 11 11
```

This repeats the printing of the loop i. Consider another example:

```r
mat=matrix(0,4,4); for(i in 1:4){for(j in 1:4){mat[i,j]=i*j}};mat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    2    4    6    8
## [3,]    3    6    9   12
## [4,]    4    8   12   16
```

This create a matrix with 4 rows and 4 columns and fill each matrix element using the product of the two loops values (i and j). Please note the use of ; . This allows separating different things you need to do in sequence. So ; allows telling RStudio, stop doing something, do something else.

Example: an important stochastic process

Consider a vector of random normal observations whose name is "e" as follows: `{r} set.seed(130);e<-rnorm(100)` Now generate the following process

$$y_t = y_{t-1} + e_t$$

for t running from 1 until 100 Finally using the function plot() show it! Can you recognize what type of process is it??

The if condition

The condition (if) makes a procedure when something happens or not (does not make it). For example, the following example generate a variable x being a random uniform distribution between 3 and 8. Secondly, if x is <6 then you will see the message x is < than 6! , otherwise you will see nothing. Try;

```r
x=runif(1,3,8);
if(x<=6) {print("x is < or equal than 6!")} else{print("x is > than 6!")}
```

```
## [1] "x is < or equal than 6!"
```

```
x
```

```
## [1] 4.445799
```

The function runif(n, min, max) generate n random uniform values between the min value and the max value. In the above example, when the number is <6 than you see the print otherwise you don't.

When you write an if condition containing different possibilities you need to start with if, continue with else if, endup with else.

Example:

```
x<-round(runif(1,0,3),0)
x
```

```
## [1] 0
```

```
if(x==0){
  print("zero")
} else if(x==1){
  print("one")
} else if(x==2){
  print("two")
    } else {print("none")}
```

```
## [1] "zero"
```

Note the use of round, approximating the number to the closer integer. For exampe if you do:

```
round(3.52456, 3)
```

```
## [1] 3.525
```

Note that you can use if several times but then you have to put ; in the middle. The chunk below is the same as the one shown above:

```
x<-round(runif(1,0,3),0)
if(x==0){
  print("zero")
}; if(x==1){
  print("one")
}; if(x==2){
  print("two")
  };if(x==3) {print("none")}
```

```
## [1] "none"
```

The following creates a vector y (of dimension 20) of random uniform values. Secondly, it creates another vector called z (of unknown dimension) and if the i-th value of y is <=0 then it assign to the relative i-th value of z the value -1. Otherwise if y is>0 it assigns to z[i] the value of 1:

```r
y=c(round(runif(20,-10,10)));z=c();for(i in 1:20){
  if(y[i]<=0){z[i]<- -1}else if(y[i]>0){z[i]<- 1}}
#Lets check if it worked...
t(cbind(y,z))
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## y    8    4    6  -10    0    5   -6   -4   -5    -7    -2    -2    -3    -7
## z    1    1    1   -1   -1    1   -1   -1   -1    -1    -1    -1    -1    -1
##   [,15] [,16] [,17] [,18] [,19] [,20]
## y    -7    -5    -1    -5     7    -9
## z    -1    -1    -1    -1     1    -1
```

Consider the following process:

`{r}set.seed(33); x<-c();x[1]<-45;for(i in 2:1000)x[i]<-x[i-1]+rnorm(1);plot(x)`

Suppose that this is the price of an asset quoted in a stock market.

Now create a binary variable taking the values:

1 if the price goes up from one day to another one

0 if the price goes down from one day to another one

Now let's make an exercise to revise.

Exercise 2

(1) Create a vector (give it the name v) having only odds values and running from 3 until 37.

(2) Using the function matrix(,,), create a vector of zeros, called z, having the same length of v.

(3) Using the for loop and the if condition, fill the vector z such that if the generic element of v is <=14 then its corresponding element in z is equal to 1. Else if the generic element of v is >14 and <=26 then its corresponding element in z is equal to 2. Finally, if the generic element of v is >26 and <=40 then its corresponding element in z is equal to 3.

(4) Consider that a random normal number can be generated using rnorm(1). Similarly, 10 random normal numbers can be generated using rnorm(10).

(5) Create a matrix (give it the name matrice having 3 rows and 20 columns and filling its elements using random normal numbers.

(6) Create a matrix, called CorrelationMatrix, having 3 rows and 3 columns with all elements equal to zero. Now, fill the matrix CorrelationMatrix such that the generic element in the ith row and jth column is the correlation between the ith row of matrice and the jth row of matrice.

Data scraping from the Web: Reading data from HTML tables (web-scraping).

Suppose you want to work with data that are available from the Web.

First download the package XML by clicking

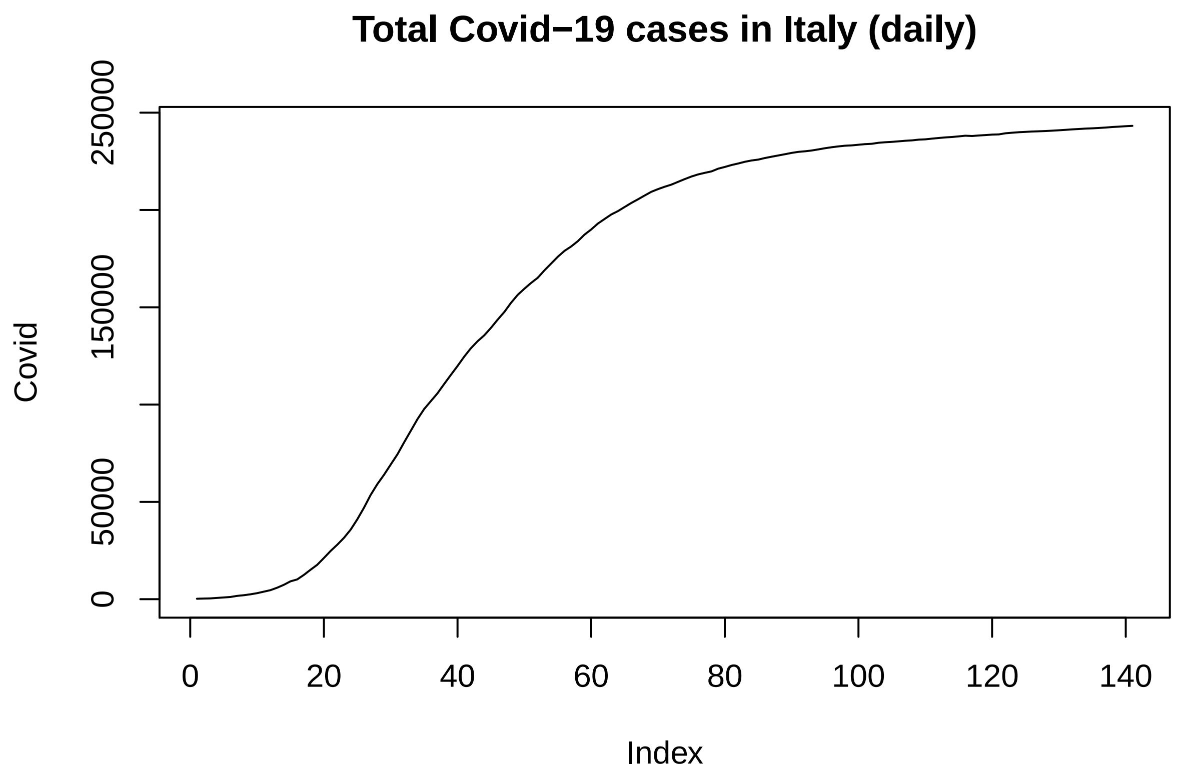
```r
# install.packages("XML")
# library(XML)
```

Suppose we want to analyze the data relative to the daily contagion of the Coronavirus (COVID19) in Italy.

The following webpage http://www.pangoo.it/coronavirus/ contains data at national level but you can also download data at regional and at city level.

Now if we want to download the data at the national level:

```r
library(XML)
theURL<-'http://www.pangoo.it/coronavirus/?t=country&r=ITA&data=y#table'
thedata<-readHTMLTable(theURL)
CovidItaly<-as.data.frame(thedata[[1]])
Totalcases<-as.matrix(CovidItaly$V8)
plot(Totalcases[1:(length(Totalcases)-1)],type="l",
     main="Total Covid-19 cases in Italy (daily)",ylab = "Covid")
```



This is just one simple example to show how to use R for data scraping. However, there are several more sophisticated packages, such as the ‘rvest’ package, that can help you make more advanced data scraping with R.

The quantmod package

An interesting package that can be used in R and Rstudio is the *quantmod* package. You can load it into Rstudio by digiting *install.package*(”*quantmod*”). This package allows you download and chart many financial products. A nice example below:

```
library(quantmod)
```

```
## Loading required package: xts
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: TTR
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

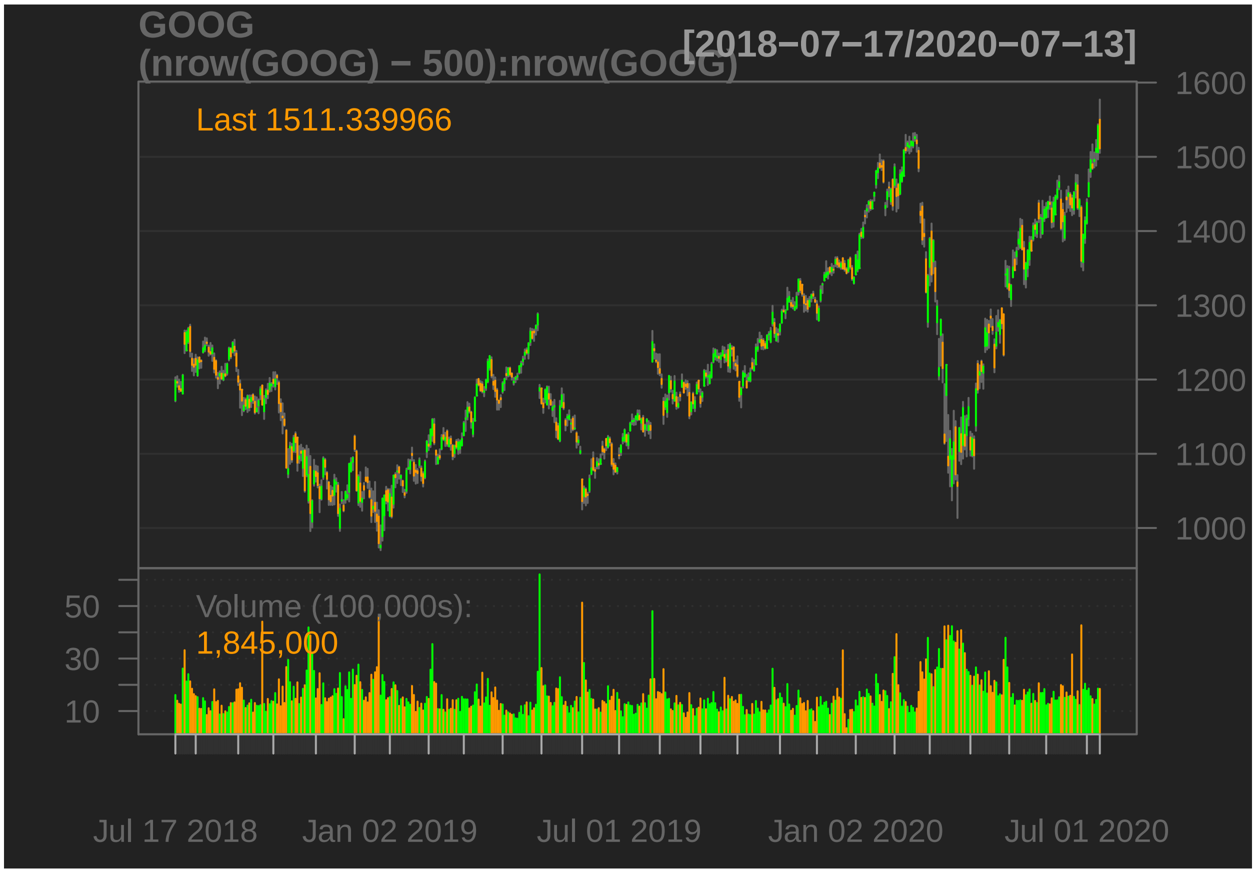
```
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```r
getSymbols(c("GOOG","GS"))
```

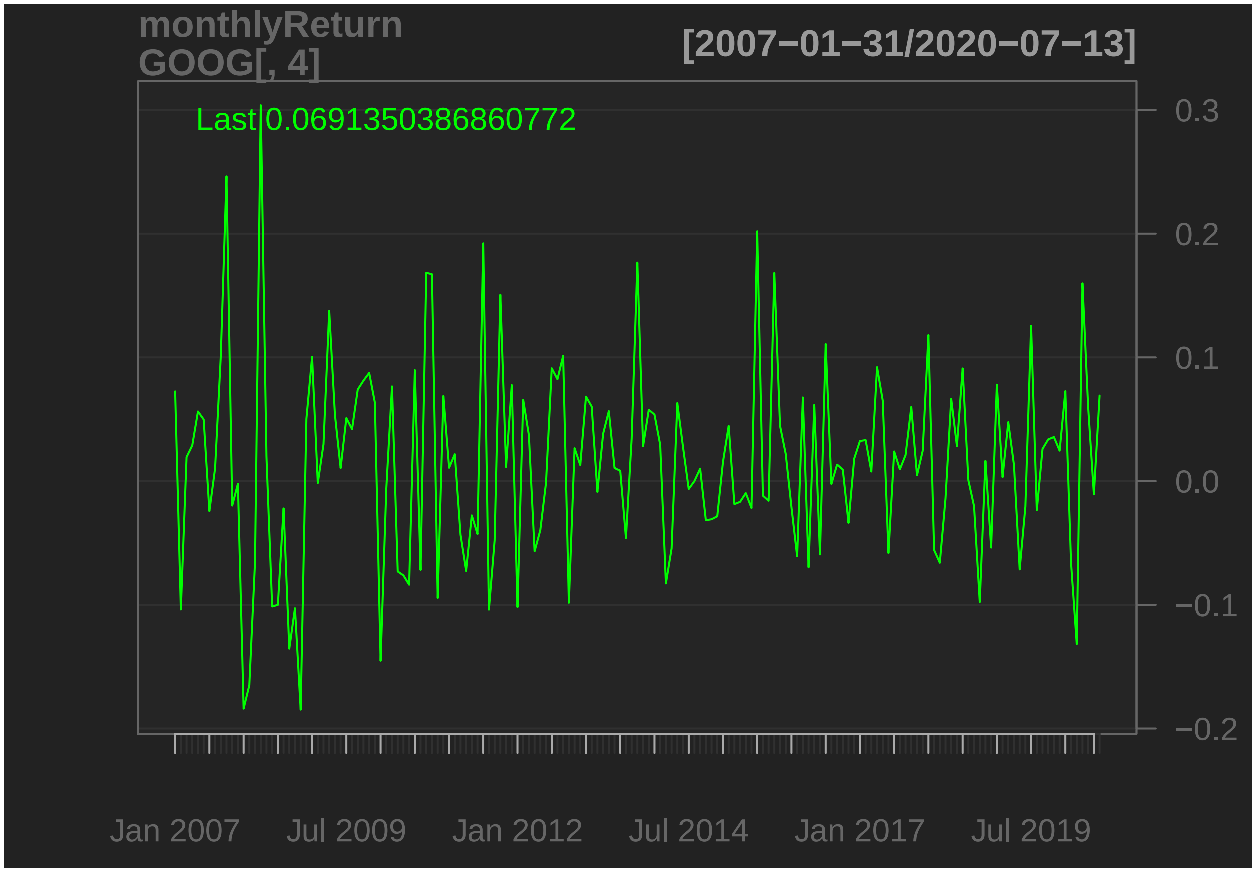
```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
## [1] "GOOG" "GS"
```

```r
chartSeries(GOOG[(nrow(GOOG)-500):nrow(GOOG),])
```

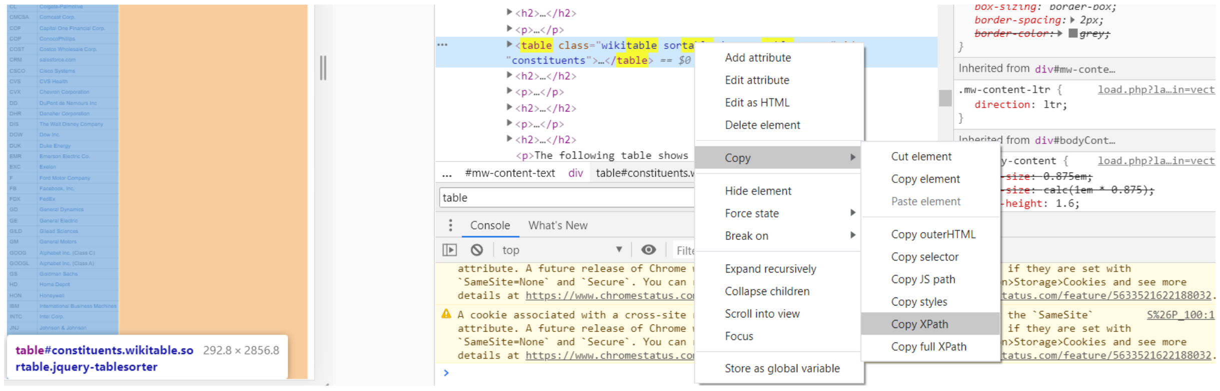


```r
chartSeries(monthlyReturn(GOOG[,4]),type="l")
```



Suppose we want to download and analyse all the stock prices of the components of the SP100, quoted in the USA. This is a nice example of webscraping that allows you tasting the power of R.

First we need the list of symbols that will be downloaded by quantmod. If you google the list you can find it in the Wikipedia page https://en.wikipedia.org/wiki/S%26P_100. Now we want to take the list as in the section "Components". We can now use of the *rvest* library (as mentioned above) as follows. Type F12 and then type on the window Element. Now you can run a search $Ctrl + F$ serching "table" . Repeat until the table on the left is highlighted in blu. Now you can copy the Xpath as in the screen below. This is what you are supposed to copy: //*[@id="constituents"]



Now in the chunk below you have the list of components of SP100 that you can download using quantmod.

```
library(rvest)
```

```
## Loading required package: xml2

##
## Attaching package: 'rvest'

## The following object is masked from 'package:XML':
##
##     xml
```

```
url<-"https://en.wikipedia.org/wiki/S%26P_100"
mylist <- url %>% read_html() %>% html_nodes(xpath='//*[@id="constituents"]') %>%
  html_table()
SP100list<-mylist[[1]]$Symbol
SP100list[19]<-"BRK-A"
SP100list
```

```
##   [1] "AAPL"  "ABBV"  "ABT"   "ACN"   "ADBE"  "AIG"   "ALL"   "AMGN"  "AMT"
##  [10] "AMZN"  "AXP"   "BA"    "BAC"   "BIIB"  "BK"    "BKNG"  "BLK"   "BMY"
##  [19] "BRK-A" "C"     "CAT"   "CHTR"  "CL"    "CMCSA" "COF"   "COP"   "COST"
##  [28] "CRM"   "CSCO"  "CVS"   "CVX"   "DD"    "DHR"   "DIS"   "DOW"   "DUK"
##  [37] "EMR"   "EXC"   "F"     "FB"    "FDX"   "GD"    "GE"    "GILD"  "GM"
##  [46] "GOOG"  "GOOGL" "GS"    "HD"    "HON"   "IBM"   "INTC"  "JNJ"   "JPM"
##  [55] "KHC"   "KMI"   "KO"    "LLY"   "LMT"   "LOW"   "MA"    "MCD"   "MDLZ"
##  [64] "MDT"   "MET"   "MMM"   "MO"    "MRK"   "MS"    "MSFT"  "NEE"   "NFLX"
##  [73] "NKE"   "NVDA"  "ORCL"  "OXY"   "PEP"   "PFE"   "PG"    "PM"    "PYPL"
##  [82] "QCOM"  "RTX"   "SBUX"  "SLB"   "SO"    "SPG"   "T"     "TGT"   "TMO"
##  [91] "TXN"   "UNH"   "UNP"   "UPS"   "USB"   "V"     "VZ"    "WBA"   "WFC"
## [100] "WMT"   "XOM"
```

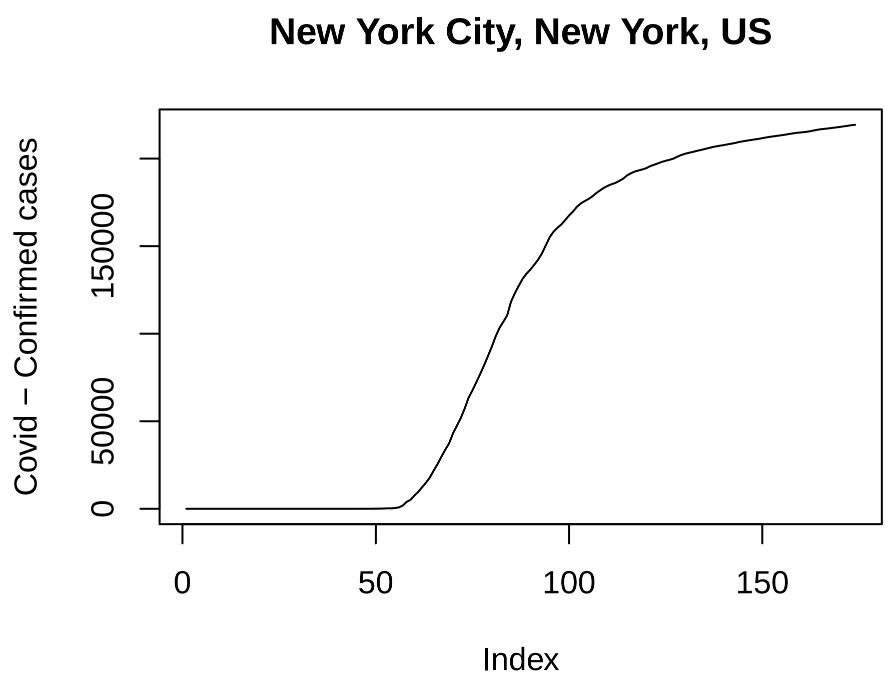
```
##The code below constructs a portfolio with the SP100 components
#getSymbols(c(SP100list))
#n<-301;portfolio<-matrix(0,n,1);for(h in 1:length(SP100list)){if(nrow(get(SP100list[[h]]))>n){portfoli

###The code below do the same as above using the SP500

#url<-"https://en.wikipedia.org/wiki/List_of_S%26P_500_companies"
#mylist <- url %>% read_html() %>% html_nodes(xpath='//*[@id="constituents"]') %>%
#  html_table()
#SPlist<-mylist[[1]]$Symbol
#SPlist[68]<-"BRK-B"
#SPlist[80]<-"BF-B"
#getSymbols(c(SPlist))
#n<-301;portfolio<-matrix(0,n,1);
#for(h in 1:length(SPlist)){
#if(nrow(get(SPlist[[h]]))>n){
#portfolio<-cbind(portfolio,get(SPlist[[h]])[(nrow(get(SPlist[[h]]))-(n-1)):nrow(get(SPlist[[h]])),4])}
```

Sometimes one can find data online with an Excel (csv) format. You can then use the *read.csv* command as follows:

```
# This is the webpage on github:
#"https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/
#/csse_covid_19_time_series/time_series_covid19_confirmed_US.csv"
covid<-read.csv("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/
csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_US.csv")
covidselection<-covid[-seq(1,11,1)] #take out the labels/names
plot(t(covidselection[1864,]),type="l",main=covid[1864,11],ylab="Covid - Confirmed cases")
```

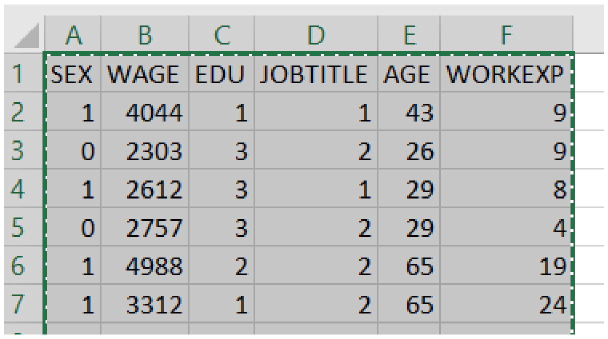


Exercise: create the series of new Covid19 cases in NY

Now suppose we want to calculate the number of new Covid cases in New York (using the series of confirmed as above). The new cases for the day t is the difference between the Confirmed cases at time t and those at time t-1. Using the chuck above calculate this series and plot it!

Using real dataset: descriptive analysis and graphics.

Opening and working with a dataset: here we show how you can open an existing dataset. Suppose you want to work with data contained in an Excel file. Then open the Excel file and select the variables you want to put in R. Now copy (Ctrl +C) the piece of information of your file Excel that you want to put in R. You should see in Excel this (see Figure 1):

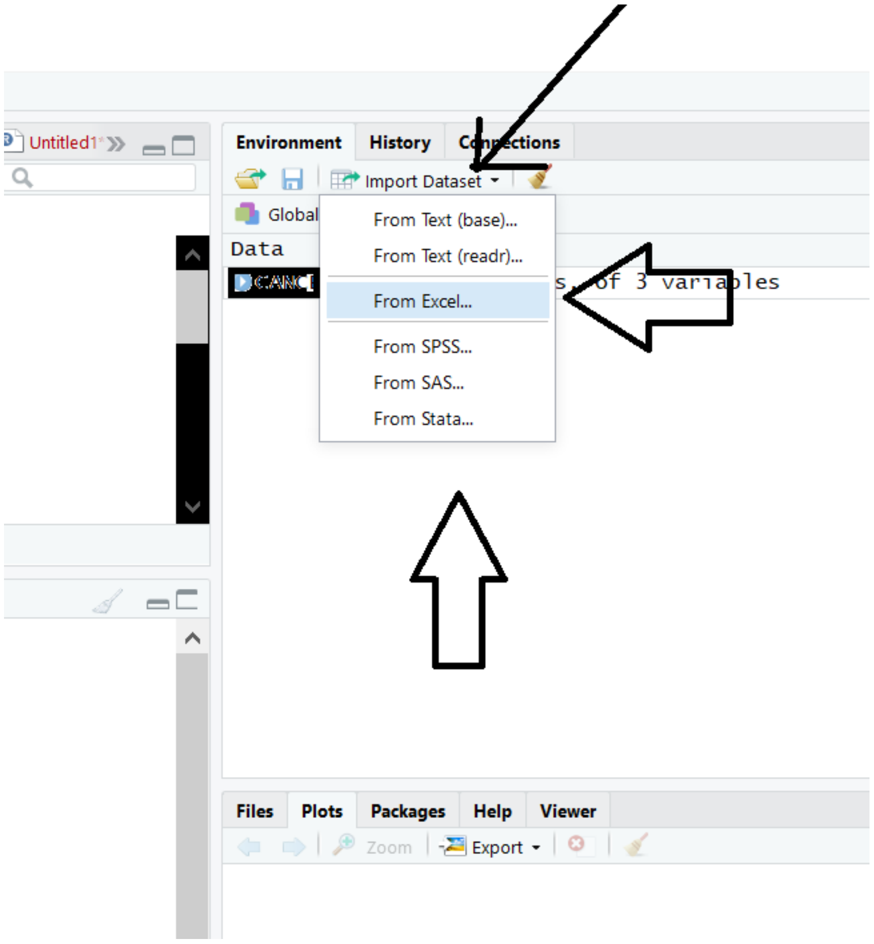


	A	B	C	D	E	F
1	SEX	WAGE	EDU	JOBTITLE	AGE	WORKEXP
2	1	4044	1	1	43	9
3	0	2303	3	2	26	9
4	1	2612	3	1	29	8
5	0	2757	3	2	29	4
6	1	4988	2	2	65	19
7	1	3312	1	2	65	24

Then go to R and digit the following command :

```
library(readxl)
#Warning! This is my working directory. You need to set your working directory if you want to upload th
setwd("C:/Users/giacomo.sbrana/OneDrive - NEOMA Business School/Documents/Liang/AppliedTimeSeries/ForStu
dataRH <- read_excel("dataRH.xlsx")
#View(dataRH)
#dataRH <- read.table("clipboard", header=TRUE)
#Remember to use TRUE after header, such that your variable name is kept.
```

Below you find another easy way to trasfert your dataset in Excel into Rstudio.



For those using the Mac please do the following:

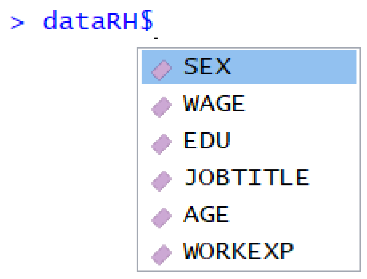
install.packages("clipr")

library(clipr)

clear_clip()

mydata<-read_clip_tbl(read_clip())

This create a dataset in R called "dataRH". By typing dataRH$ allows you to see the variables in the data set which you can select (see Figure 3).



```
> dataRH$
SEX
WAGE
EDU
JOBTITLE
AGE
WORKEXP
```

If you want to have a look only the first 6 rows of your dataset you can type:

```r
head(dataRH)
```

```
## # A tibble: 6 x 6
##     sex workexp   age jobtitle  wage   edu
##   <dbl>   <dbl> <dbl>    <dbl> <dbl> <dbl>
## 1     1       9    39        1  2370     1
## 2     1      16    31        1  3125     2
## 3     0       8    53        2  2620     1
## 4     1       6    35        1  2270     1
## 5     0       7    38        2  3315     3
## 6     0       8    59        3  3375     3
```

This allows to have a quick look at the variables. Similarly, if you want to highlight only the last 6 rows of your dataset you can type:

```r
tail(dataRH)
```

```
## # A tibble: 6 x 6
##     sex workexp   age jobtitle  wage   edu
##   <dbl>   <dbl> <dbl>    <dbl> <dbl> <dbl>
## 1     1       8    40        1  1910     1
## 2     1      10    36        1  2540     1
## 3     1       1    27        1  2345     1
## 4     1       2    43        1  2510     1
## 5     1       3    34        1  2255     1
## 6     1       4    30        1  2535     1
```

Let us now investigate the data we are dealing with. Please note that the variables refer to each individual worker of a firm. These are the variables:

Variable	Description
sex	0 if the worker is female, 1 if the worker is male
wage	The monthly wage in Euro of the worker
job title	1 is administrative 2 is a professional 3 is a manager
edu	1 undergraduate 2 postgraduate 3 doctoral degree
work experience	Number of years the worker spent in the firm
job title	The age of the worker

The command:

```r
summary(dataRH)
```

```
##       sex           workexp           age           jobtitle          wage
##  Min.   :0.000   Min.   : 1.00   Min.   :27.0   Min.   :1.00   Min.   :1890
##  1st Qu.:0.000   1st Qu.: 6.00   1st Qu.:37.0   1st Qu.:1.00   1st Qu.:2555
##  Median :1.000   Median :10.00   Median :46.0   Median :2.00   Median :2950
##  Mean   :0.576   Mean   :10.02   Mean   :45.5   Mean   :1.74   Mean   :3175
##  3rd Qu.:1.000   3rd Qu.:13.00   3rd Qu.:52.0   3rd Qu.:2.00   3rd Qu.:3550
##  Max.   :1.000   Max.   :30.00   Max.   :70.0   Max.   :3.00   Max.   :7465
##       edu
##  Min.   :1.000
##  1st Qu.:1.000
##  Median :1.000
##  Mean   :1.548
##  3rd Qu.:2.000
##  Max.   :3.000
```

this provides you with some basic statistics about the variables of the dataset.

Suppose you want to build a bivariate table considering two variables. Then if you type:

```
table(dataRH$sex,dataRH$edu)
```

```
##
##       1   2   3
##   0  93  85  34
##   1 192  71  25
```

this provides with the level of education by sex. You can modify a variable by cutting it in different slices (trances). The cut function in R allows this. For example:

```
table(dataRH$sex,cut(dataRH$wage/1000,4))
```

```
##
##     (1.88,3.28] (3.28,4.68] (4.68,6.07] (6.07,7.47]
##   0         109          85          14           4
##   1         213          55          20           0
```

The previous command split the variable WAGE in 4 different slices. If you want to construct different tables you can use the function xtabs. For example:

```
xtabs(~dataRH$sex+cut(dataRH$wage/1000,4)+dataRH$edu)
```

```
## , , dataRH$edu = 1
##
##           cut(dataRH$wage/1000, 4)
## dataRH$sex (1.88,3.28] (3.28,4.68] (4.68,6.07] (6.07,7.47]
##          0          74          13           5           1
##          1         175           7          10           0
##
## , , dataRH$edu = 2
##
##           cut(dataRH$wage/1000, 4)
## dataRH$sex (1.88,3.28] (3.28,4.68] (4.68,6.07] (6.07,7.47]
##          0          26          51           5           3
##          1          33          30           8           0
##
## , , dataRH$edu = 3
##
##           cut(dataRH$wage/1000, 4)
## dataRH$sex (1.88,3.28] (3.28,4.68] (4.68,6.07] (6.07,7.47]
##          0           9          21           4           0
##          1           5          18           2           0
```

This provides with 3 different tables that is the bivariate table sex vs. wage for the three different level of education.

You can see statistics by aggregating across a specific factor. For example:

```
aggregate(dataRH[-c(1)],by=list(gender=dataRH$sex),FUN="mean")
```

```
##   gender   workexp      age jobtitle     wage      edu
## 1      0 11.136792 45.53774 1.976415 3386.061 1.721698
## 2      1  9.194444 45.46875 1.565972 3019.427 1.420139
```

This creates the mean of all variables in data RH by sex group. Note that I deleted the first column of the data frame (in order to exclude sex from the table).

Another useful function is the one that allows to subset the data.

```
summary(subset(dataRH,subset = wage>5000))
```

```
##       sex          workexp           age           jobtitle         wage
##  Min.   :0.00   Min.   :14.00   Min.   :36.00   Min.   :2.00   Min.   :5005
##  1st Qu.:0.00   1st Qu.:17.75   1st Qu.:48.75   1st Qu.:3.00   1st Qu.:5189
##  Median :0.00   Median :23.00   Median :53.50   Median :3.00   Median :5658
##  Mean   :0.45   Mean   :22.70   Mean   :54.05   Mean   :2.85   Mean   :5727
##  3rd Qu.:1.00   3rd Qu.:27.25   3rd Qu.:60.75   3rd Qu.:3.00   3rd Qu.:5944
##  Max.   :1.00   Max.   :30.00   Max.   :68.00   Max.   :3.00   Max.   :7465
##       edu
##  Min.   :1.0
##  1st Qu.:1.0
##  Median :1.0
##  Mean   :1.4
##  3rd Qu.:2.0
##  Max.   :2.0
```

This table is different from the one considering all people summary(dataRH) since it considers only those with wage>5000.

Exercise 3:

Provide the proportion of female and men for:

(1) Those having a wage <3000

(2) Those having a wage >3000 and <6000

(3) Those having a wage >6000

Considering the dataset dataRH. Suppose we want to create a variable grouping the variable dataRH$age in three different group.

First create a new variable (vector), call it agegroup, having the same length than the other variables.

Using the "for" loop, modify the variable agegroup assigning the value 1 when the worker is younger than 35 years old, 2 for those between 35 and 55 and 3 for those older than 55.

In addition create a new variable (vector), call it wagegroup, having the same length than the other variables.

Using the "for" loop, modify the variable wagegroup assigning the value 1 when the worker has a wage<2500 Euro, 2 for those between having a salary between 2500 and 3500, 3 for those having a wage between 3500 and 4500, 4 for those with a wage>4500.

Please describe the social and economic aspects of the workers working in this firm by answering the following questions:

What are the variables that affect the wage of workers?

What is the sex having the highest education level?

Considering the variable jobtitle is equal to 1 if the worker is administrative, 2 if the worker is a professional, 3 if the worker is a manager. What is the proportion of female and male into these three categories?

You have noted that in the exercise above we have created new variables that were not included in the dataset we copied and pasted from an external source. Suppose that now you want to include your new variables in the dataset ":dataRH". Since your new variable "agegroup" is basically a vector then you can bind the dataset and the vector as follows:

```
#dataRH2<-cbind(dataRH,agegroup)
```

Now if you want to check that this is a new dataset also containing you new variable agegroup then type:

```
#head(dataRH2)
```

So the command cbind() concatenates the dataset with the new vector.

Please not that a similar procedure can be run if you want to concatenate two or more rows. In this case you have to use rbind(,). This will do the same but using rows.

Testing independence between two (qualitative) variables

Remember that the Null-hypothesis (H0 hypothesis) is that the two variables are independent. The alternative hypothesis (H1 hypothesis) is that there is dependence between the two variables.

First create the bivariate table (contingency table) then you insert it into the chisq.test function in R. This is the procedure:

```
mytable=table(dataRH$sex,dataRH$jobtitle)
chisq.test(mytable)
```

```
##
##  Pearson's Chi-squared test
##
## data:  mytable
## X-squared = 200.38, df = 2, p-value < 2.2e-16
```

You can see the statistics together with the degrees of freedom and the p-value.

When the p-value is >0.05 than we accept the Null-hypothesis to conclude that there is independence between the two variable

When the p-value is <0.05 than we reject the Null-hypothesis to conclude that there is dependence between the two variable

Consider this example:

```
datagenerate<-matrix(round(runif(500,1,4),0),250,2)
mytable=table(datagenerate[,1],datagenerate[,2])
chisq.test(mytable)
```

```
##
##  Pearson's Chi-squared test
##
## data:  mytable
## X-squared = 13.794, df = 9, p-value = 0.1298
```

Are you able to interpret it? Both the way I generate the data and the results?

Now consider another example:

```
datagen1<-matrix(round(runif(500,1,4),0),500,1)
datagen2<-datagen1+matrix(1,500,1)
mytable=table(datagen1,datagen2)
chisq.test(mytable)
```

```
##
##  Pearson's Chi-squared test
##
## data:  mytable
## X-squared = 1500, df = 9, p-value < 2.2e-16
```

Are you able to interpret it? Both the way I generate the data and the results?

Testing the difference between two means (quantitative variables)

Remember that the Null-hypothesis (H0 hypothesis) is that the difference between the two means is not significantly different. The alternative hypothesis (H1 hypothesis) is that the two means are significantly different.

```r
t.test(dataRH$wage[dataRH$sex==0], dataRH$wage[dataRH$sex==1])
```

```
##
##  Welch Two Sample t-test
##
## data:  dataRH$wage[dataRH$sex == 0] and dataRH$wage[dataRH$sex == 1]
## t = 4.6984, df = 449.1, p-value = 3.491e-06
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  213.2779 519.9905
## sample estimates:
## mean of x mean of y
##  3386.061  3019.427
```

You can see the statistics together with the degrees of freedom and the p-value.

When the p-value is >0.05 than we accept the Null-hypothesis to conclude that there is no significant difference between the two means

When the p-value is <0.05 than we reject the Null-hypothesis to conclude that there is a significant difference between the two means.

Consider this example:

```r
WageMenFemale<-matrix(2000+25*rnorm(500,1,4),250,2)
t.test(WageMenFemale[,1], WageMenFemale[,2])
```

```
##
##  Welch Two Sample t-test
##
## data:  WageMenFemale[, 1] and WageMenFemale[, 2]
## t = -0.71385, df = 497.48, p-value = 0.4757
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -24.95187  11.65249
## sample estimates:
## mean of x mean of y
##  2023.697  2030.347
```

Are you able to interpret it? Both the way I generate the data and the results?

Now consider another example:

```r
WageMen<-matrix(2500+20*rnorm(500,1,4),250,2)
WageFem<-matrix(2000+15*rnorm(500,1,4),250,2)
t.test(WageMen, WageFem)
```

```
##
##  Welch Two Sample t-test
##
## data:  WageMen and WageFem
## t = 113.57, df = 953.76, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  493.7709 511.1357
## sample estimates:
## mean of x mean of y
##  2521.062  2018.608
```

Are you able to interpret it? Both the way I generate the data and the results?

Exercise 4:

Consider the dataset dataRH.

Create the bivariate table between sex and edu.

Create a similar table showing the proportions such that the total must sum up to 1.

Test if there is independence between the level of education and the gender.

What do you conclude?

What is the average age of female? In addition, that of male?

Test if there is a significant different between the average ages of female and male.

What do you conclude?

What have you learnt during this class?

1) ; tells RStudio that you stop doing something

2) How to use the for loop. Example: x=c();for(i in 1:8){x[i]=i}

3) How to use the if condition. Example: g<-c();for(i in 1:8){if(x[i]<=4){g[i]=-5}else if(x[i]>4){g[i]=3}}

4) The function round approximate the values to the closest integer

5) The function rnorm(n) generate a series of n random normal number (mean 0 and var=1)

6) The function runif(n,min=i,max=j) generate a series of n random uniform numbers between the minimum i and the maximum j

7) Two way to copy your dataset (in Excel or txt) and paste it in R:

8) Firstly, open your Excel file, select the data and copy them. Secondly, type namedataset <- read.table("clipboard", header=TRUE)

9) If you have a txt file namedataset <- read.table("yourtxtfile.txt", header=TRUE)

10) head() #gives the first 6 rows of a large vector, matrix etc.

11) tail() #gives the last 6 rows of a large m vector, matrix etc.

12) summary # get the summary statistics of a single variable, or of all variables in a dataset

13) table # frequency counts of entries of two variables.

14) cut # divides the range of x into intervals and codes the values in x according to which interval they fall.

15) xtabs()# this function is similar to table but.consider this example: xtabs(~VAR1+ VAR2+ VAR3) this corresponds to create the table(VAR1,VAR2) for each value of VAR3.

16) >= #this is bigger or equal than

17) <= #smaller or equal than

18) & # and

19) | # or

20) == # this specifies that you refer to a specific value. Used for example when using the if condition.

21) install.packages("thepackagename") # this install a specific R package. Example the plot3Drgl is a package allowing for plotting 3D graphics

22) library("thepackagename") #this open the package to be used during a session.

23) aggregate #splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form. Example: aggregate(dataset,by=list(something=x),FUN="mean")

24) cbind(x,y) #concatenates two vectors x and y or two matrix. Note that you should have the same number of rows

25) rbind(x,y) #concatenates two row vectors x and y or two matrix. Note that you should have the same number of columns.

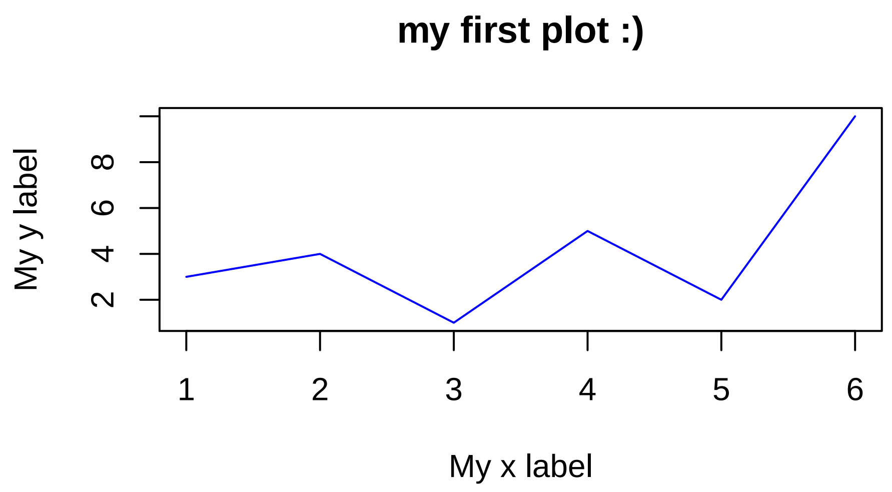
Data visualization

A quick introduction

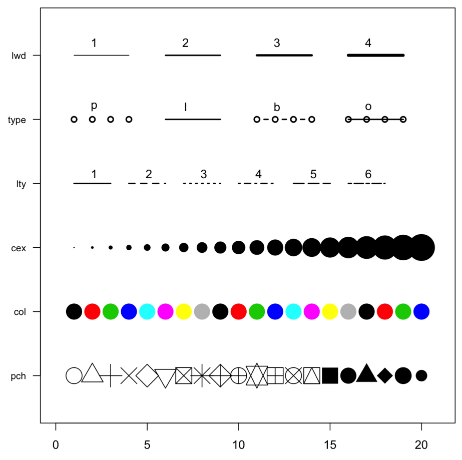
We now step into the world of graphical respresentation with R. The material provided is an introduction about the potentials of R for data visualization. However, if you want to further increase your knowledge, you can find many R packages online that help you making amizing graphics.

The simplest and easier plot in R can be shown as follows:

```r
y<-c(3,4,1,5,2,10) # this is a vector
plot(y,type="l",main="my first plot :)",xlab = "My x label",ylab = "My y label",col="blue")
```

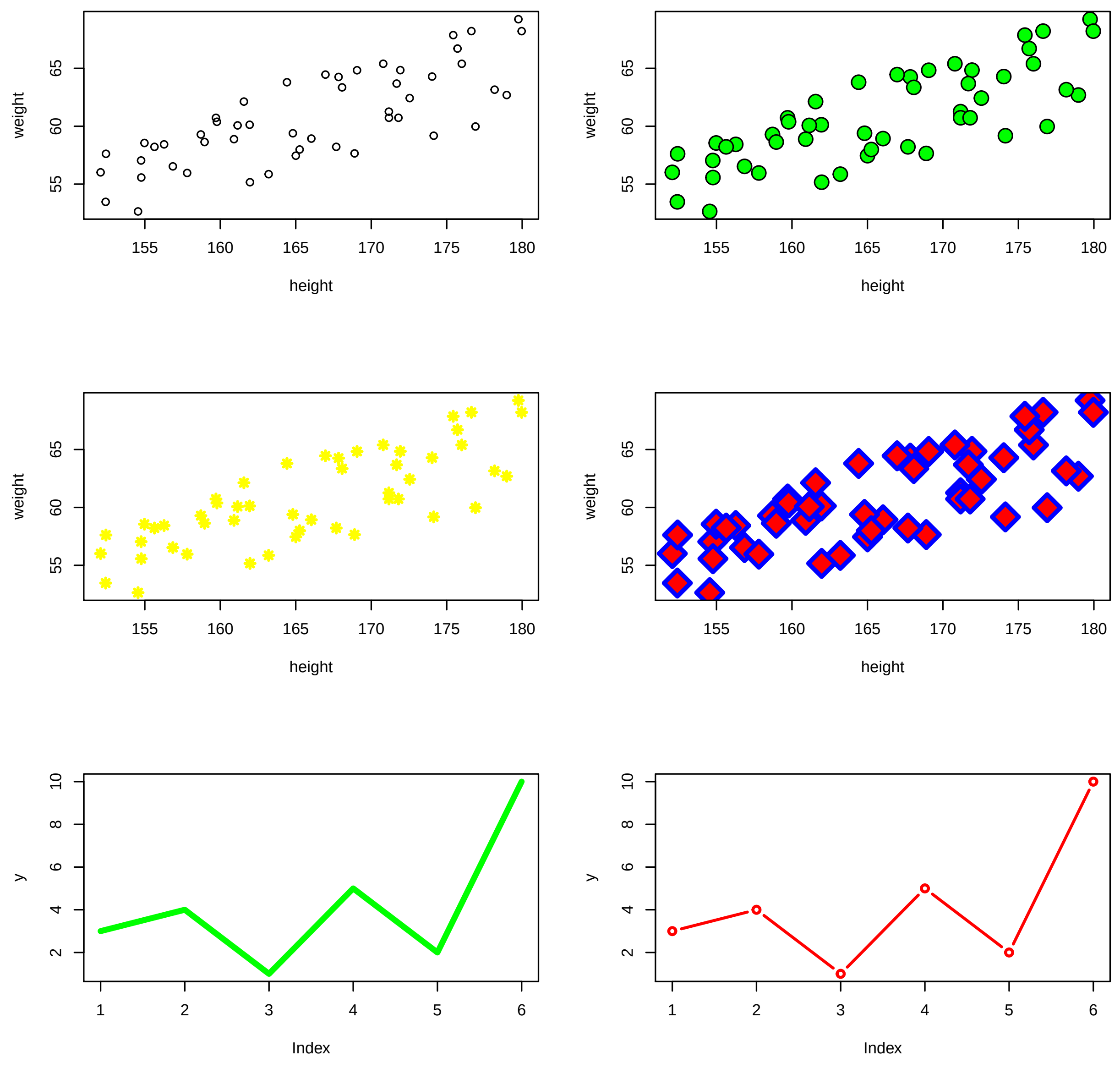


In the squared Figure you can find some interesting details you can use to improve your graphical aspects:



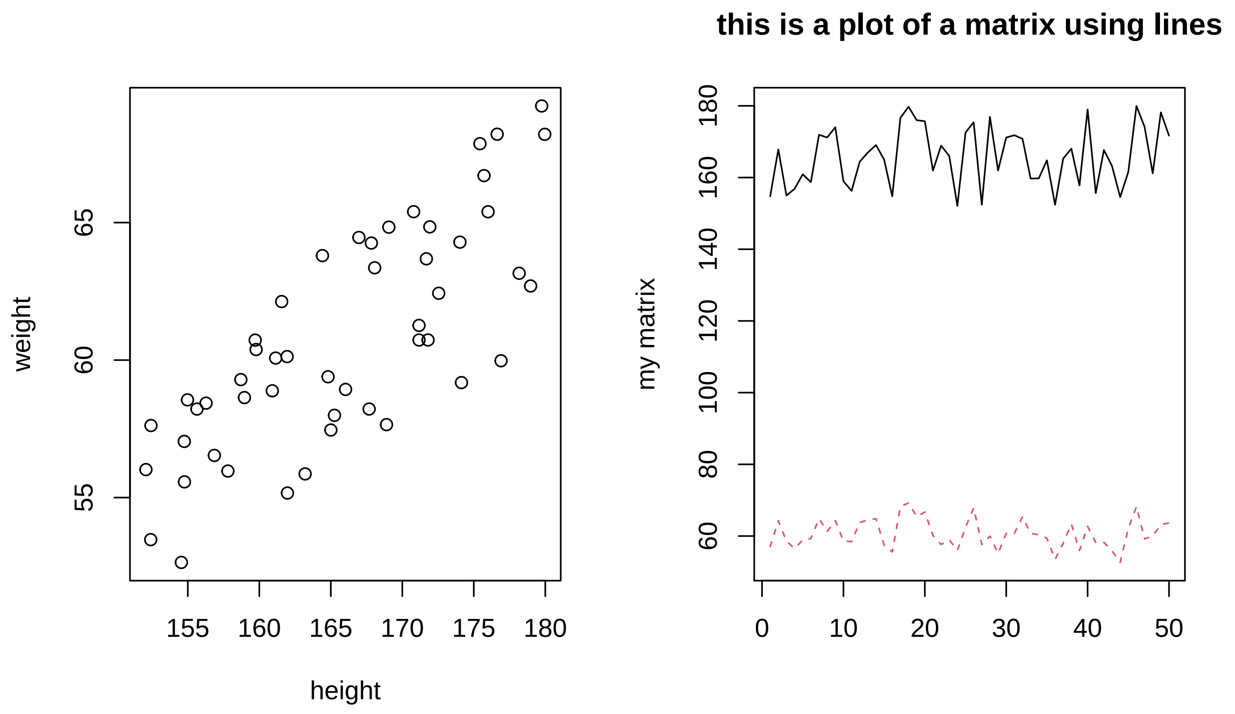
Consider the folliwing example such as we have 50 students providing their height and weight. Data are generated in the following chunck of code.

```r
n<-50
height<-matrix(150+runif(n,1,30),n,1)
weight<-matrix(height/3+runif(n,1,10),n,1)
par(mfrow=c(3,2))#this option allows to put 6 graphics in 3 rows and 2 columns
plot(height,weight)
plot(height,weight,pch = 21, cex=2, col="black", bg="green", lwd=1)
plot(height,weight,pch = 8, cex=1, col="yellow", lwd=2)
plot(height,weight,pch = 23, cex=3, col="blue", bg="red", lwd=3)
plot(y,lwd=4,col="green",type="l")
plot(y,lwd=2,col="red",type="b")
```



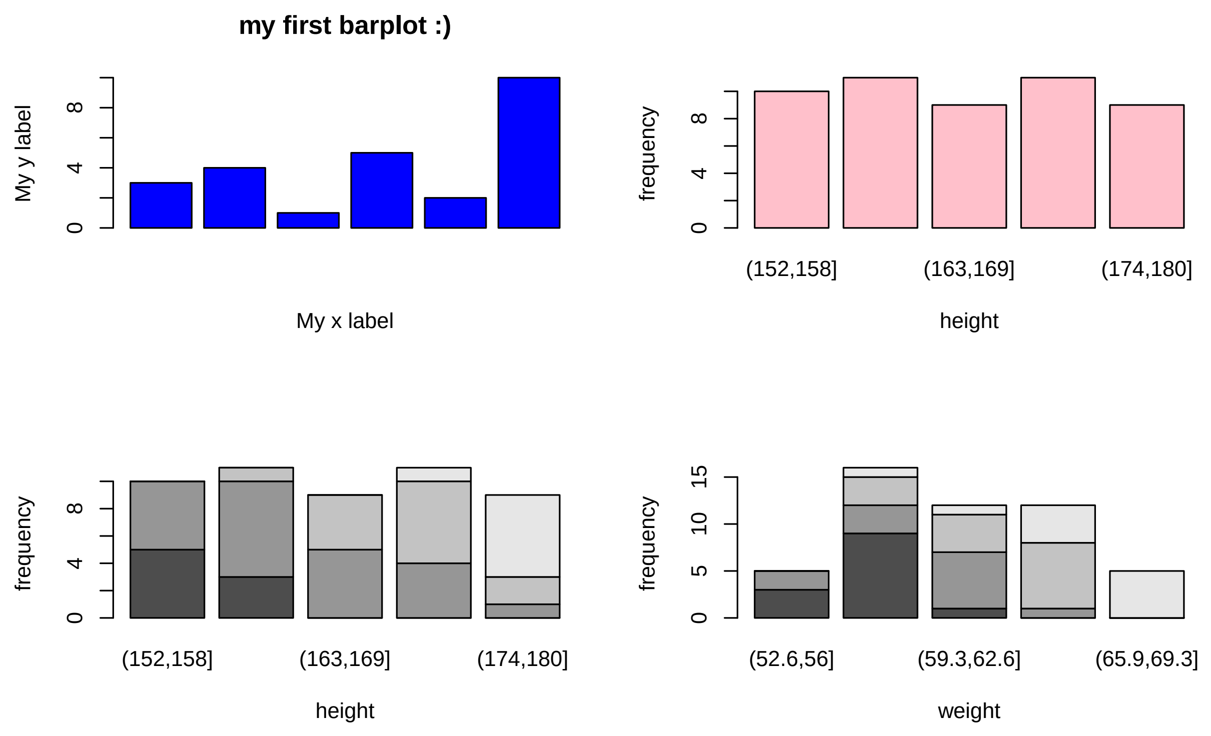
Using the information as in Figure 1 you can figure it out why we obtain the shown output. Suppose you have a matrix and you want to assign name to the columns. You can do the same as before as follows:

```
both<-cbind(height,weight)
colnames(both)<-c("height","weight")
par(mfrow=c(1,2))
plot(both)
matplot(both,type = "l",main="this is a plot of a matrix using lines",ylab = "my matrix")
```



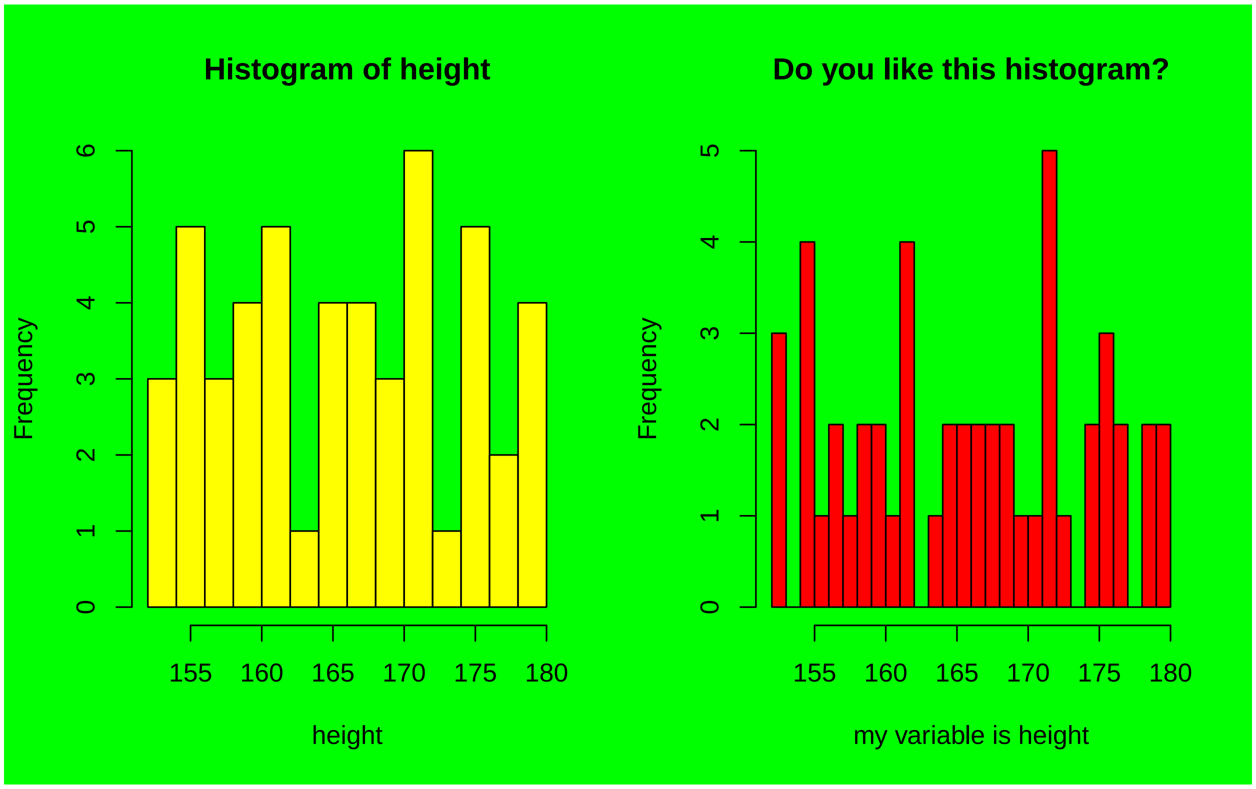
Note however that in one case the we plot the values of the matrix both in a two dimensional graphic, while in the other case we plot the two vectors as separated. Here below we introduce the barplot:

```
y<-c(3,4,1,5,2,10) # this is a vector
bothframe<-data.frame(height,weight)
par(mfrow=c(2,2))
barplot(y,main="my first barplot :)",xlab = "My x label",ylab = "My y label",col="blue")
barplot(table(cut(bothframe$height,5)),co="pink",xlab = "height",ylab = "frequency")
barplot(table(cut(bothframe$weight,4),cut(bothframe$height,5)),xlab = "height",ylab = "frequency")
barplot(table(cut(bothframe$height,4),cut(bothframe$weight,5)),xlab = "weight",ylab = "frequency")
```



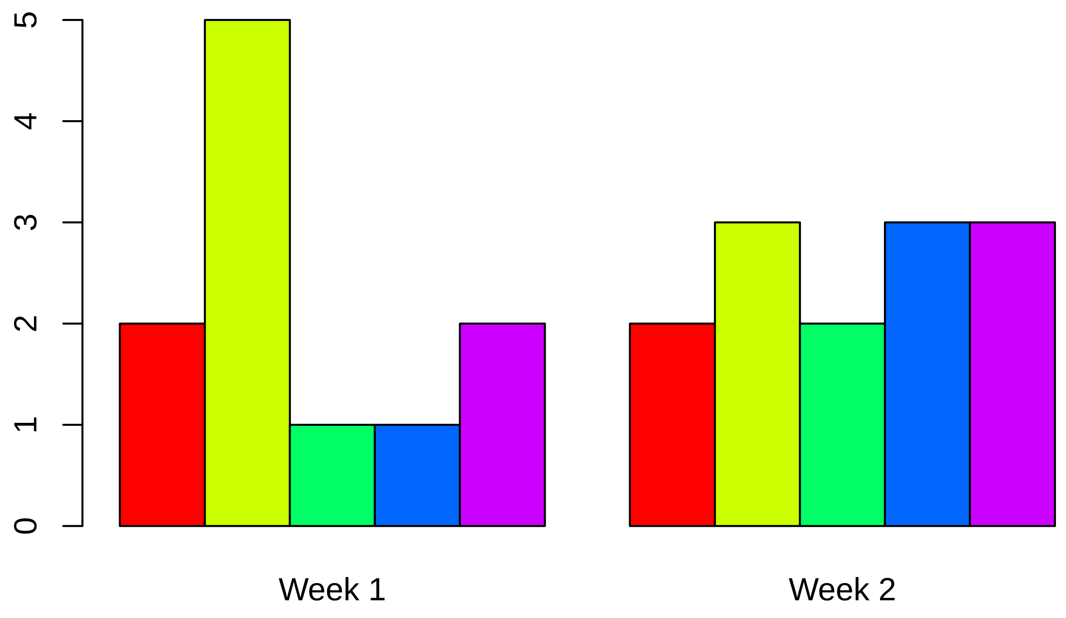
Here we introduce the histogram plot.

```
par(mfrow=c(1,2),bg="green")
hist(height,col="yellow",breaks = 10)
hist(height,breaks = 30,xlab = "my variable is height",main="Do you like this histogram?",col="red")
```



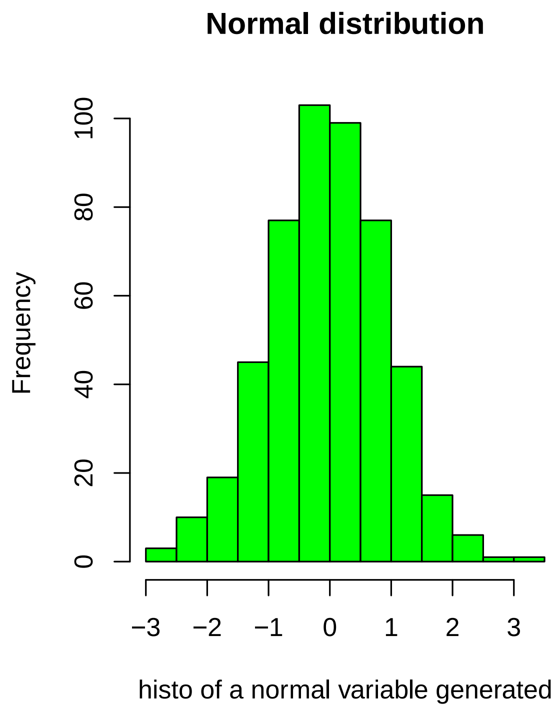
Suppose you want to plot the data of a matrix. You need to give the columns a name as below then you can see and compare the values using the barplot.

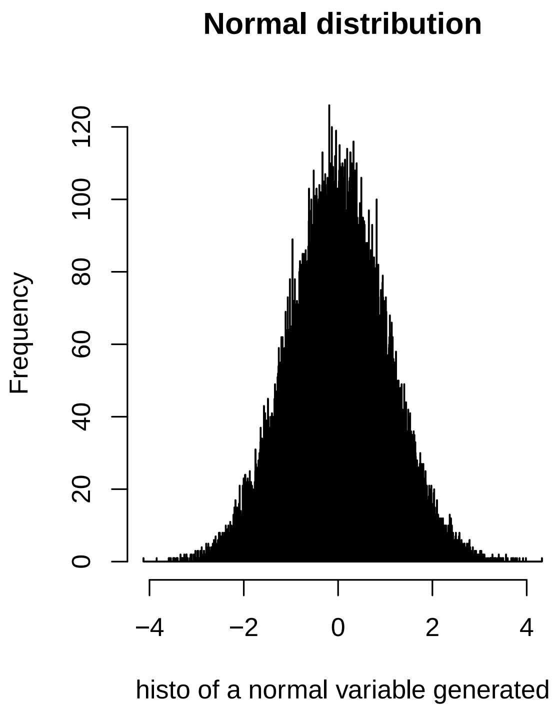
```r
dati<-matrix(round(runif(10,1,5),0),5,2)
colnames(dati)<-c("Week 1","Week 2")
barplot(dati,   beside=TRUE, col=rainbow(5))
```



You might increase the number of bars (breaks) in the histogram as in the example below where the higher the breaks the more precise the normal distribution is approximated.

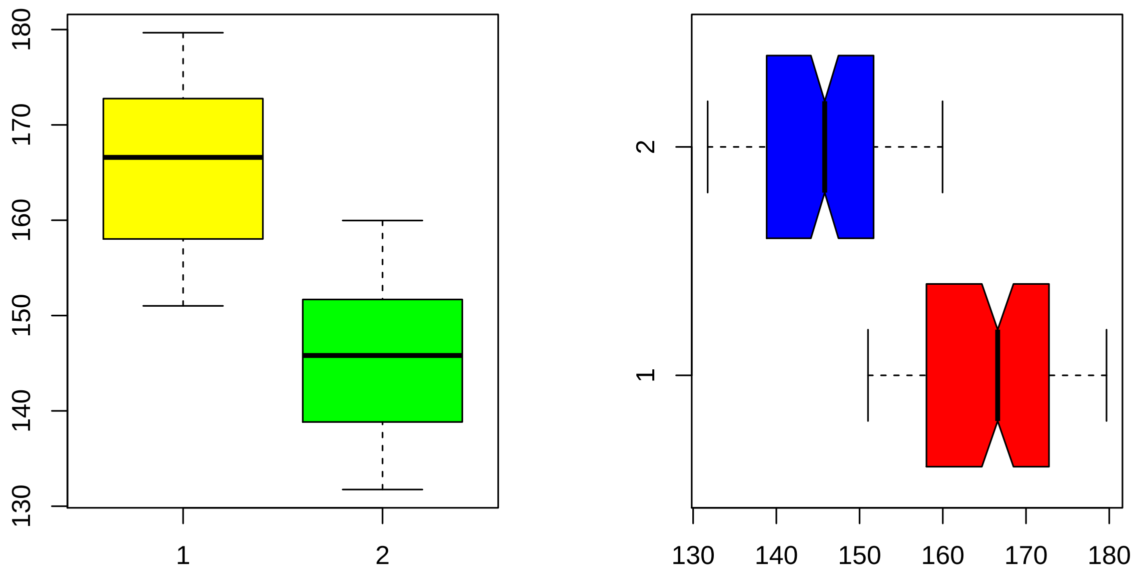
```r
par(mfrow=c(1,2))
hist(rnorm(500),breaks = 20,
     xlab = "histo of a normal variable generated",
     main="Normal distribution",col="green")
hist(rnorm(50000),breaks = 2000,
     xlab = "histo of a normal variable generated",
     main="Normal distribution")
```





Another important reprentation is the boxplot that allows showing the interval in which values are concentrated. Below you see them in horizontal and vertical positions.

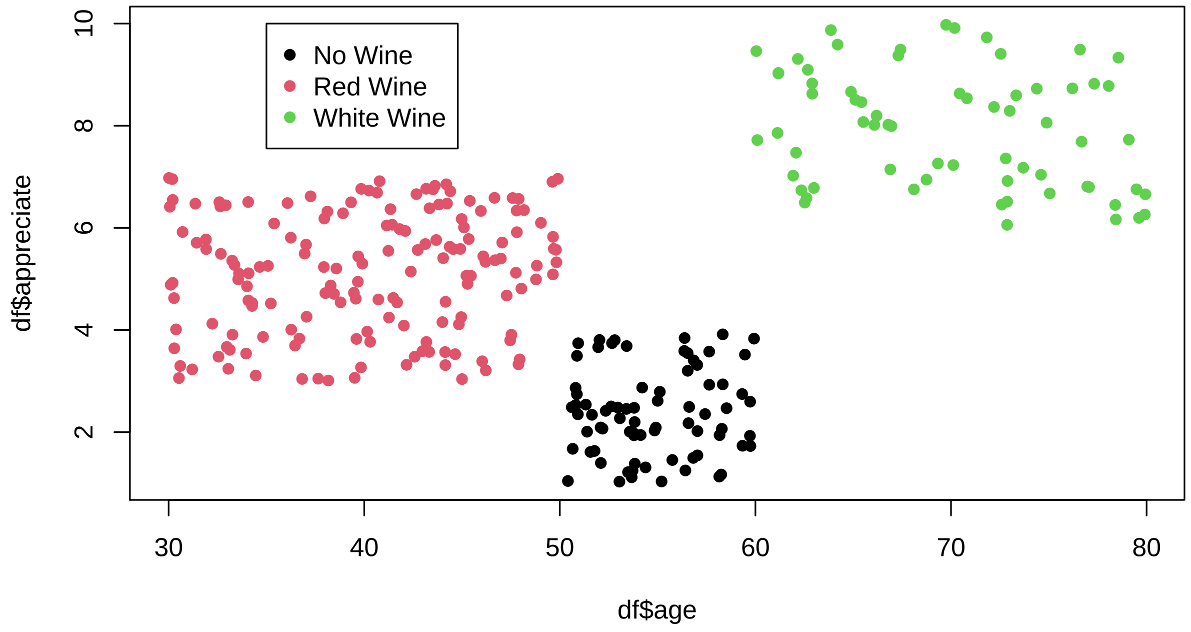
```
n<-150
height1<-matrix(150+runif(n,1,30),n,1)
height2<-matrix(130+runif(n,1,30),n,1)
par(mfrow=c(1,2))
boxplot(cbind(height1,height2),col=c("yellow","green"))
boxplot(cbind(height1,height2),horizontal=TRUE, col=c("red","blue"),notch=T)
```



Sometimes it is useful to put more variables is a graphic. For example. Suppose that you sell cheese and you want to analyze the impact of drinking (or not) wine on cheese appreciation. Therefore you ask a group of 300 people how much they like a specific type of cheese while they are (1) Not drinking wine (2) Drinking White wine (3) Drinking Red wine. So you create a factor identifying if the person who tasted cheese was drinking or not. secondly you record the level of appreciation of cheese together with the age of the person. Suppose the appreciation is between 0 and 10. A simulation with results are reported below. Here the graphical representation is especially useful since we see the cross plot between age and appreciation by the factor type (Drinking or not wine).

```r
facts = matrix(round(runif(300,1,3),0),300,1)
myfactor = factor(facts,labels=c("No Wine","Red Wine","White Wine"))
appreciate<-c();
for(i in 1:300){
  if(myfactor[i]=="No Wine"){
    appreciate[i]<-runif(1,1,4)
  } else if(myfactor[i]=="Red Wine"){
      appreciate[i]<-runif(1,3,7)
  } else if(myfactor[i]=="White Wine"){
        appreciate[i]<-runif(1,6,10)}
  }
age<-c(); for(i in 1:300){
  if(myfactor[i]=="No Wine"){
    age[i]<-runif(1,50,60)
  } else if(myfactor[i]=="Red Wine"){
      age[i]<-runif(1,30,50)
  } else if(myfactor[i]=="White Wine"){
        age[i]<-runif(1,60,80)}
  }
df<-data.frame(appreciate,age,myfactor)
```

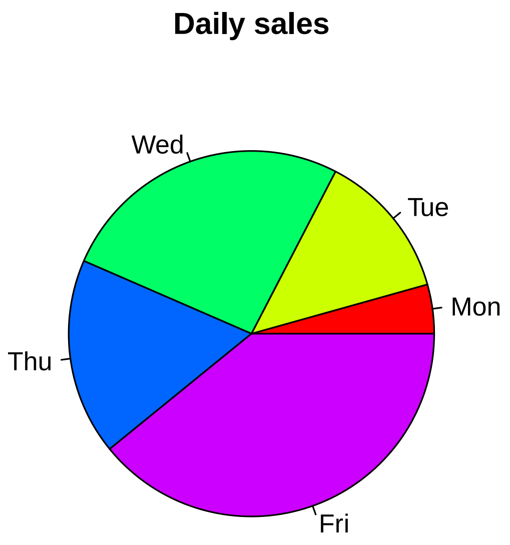
```r
plot(df$age,df$appreciate,col=df$myfactor,pch = 16)
legend(x = 35, y = 10, legend = levels(df$myfactor), col = c(1:3), pch = 16)
```



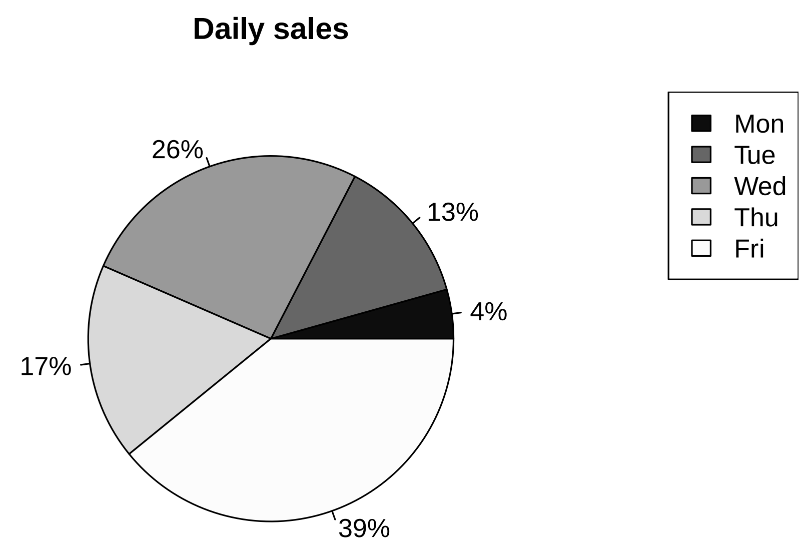
Clearly we can see that people that appreciated the most are the oldest one and they like to drink White wine. This is an important marketing information. Those who do not drink wine they do not like cheese (and they are aged between 50 and 60).

Below you can see how to make a pie.

```r
sales <- c(1, 3, 6, 4, 9)
factorsales<-factor(sales,labels =c("Mon","Tue","Wed","Thu","Fri"))
pie(sales, main="Daily sales", col=rainbow(length(sales)),
   labels=c("Mon","Tue","Wed","Thu","Fri"))
```



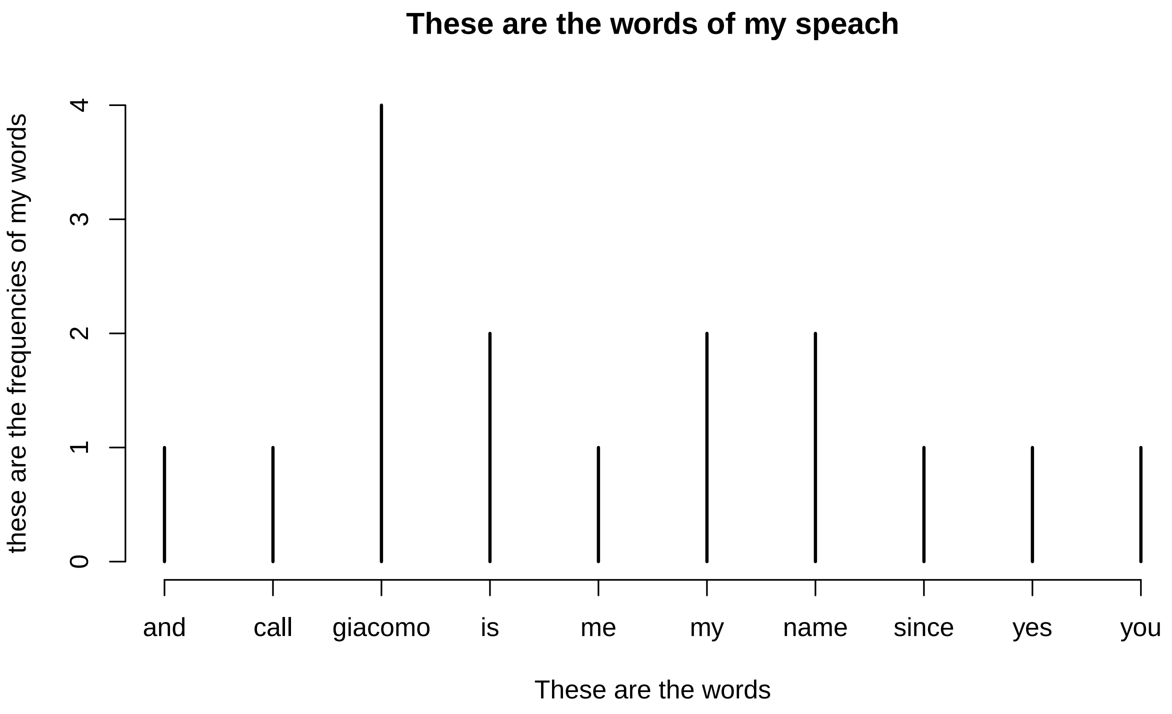
```r
propor<-round(sales/sum(sales),2)
colors=c("grey5","grey40","grey60", "grey85","grey99")
pie(sales,labels=paste(round(sales/sum(sales),2)*100,"%",sep=""), main="Daily sales", col = colors)
legend("topright", legend = levels(factorsales), fill = colors)
```



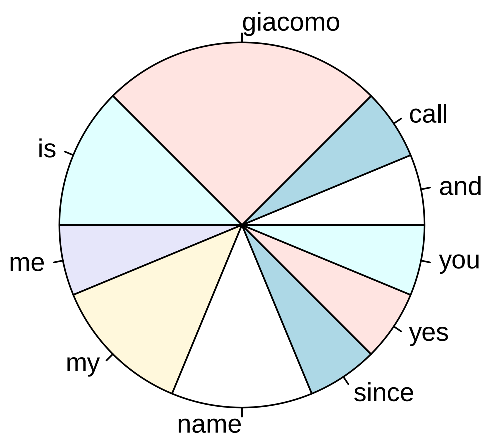
Text analysis? A simple example to visualize words!

Suppose you listen to me and you hear this:

```
myspeech<-"giacomo is my name yes giacomo and you call me giacomo since giacomo is my name"
vectormyspeech<-unlist(strsplit(myspeech,split= " "))
plot(table(vectormyspeech),main="These are the words of my speach",
     ylab="these are the frequencies of my words",xlab = "These are the words")
```

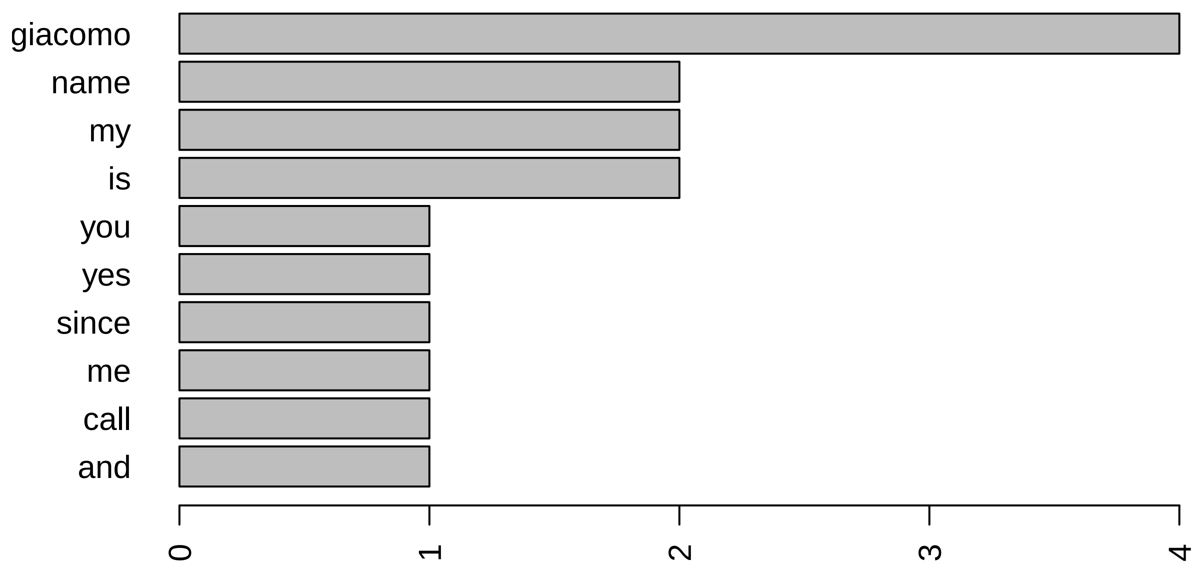


```
pie(table(vectormyspeech))
```



This is another way to show the most frequent words pronounced:

```r
barplot(sort(table(vectormyspeech)),horiz = TRUE,las=2)
```

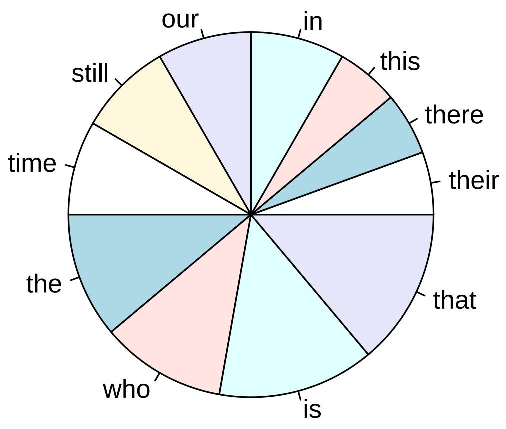


Below the victory speech made by Obama in 2008

```r
a1<-"If there is anyone out there who still doubts that America"
a2<-"is a place where all things are possible who still wonders"
a3<-"if the dream of our founders is alive in our time who still questions"
a4<-"the power of our democracy tonight is your answer It is the answer"
a5<-"told by lines that stretched around schools and churches in numbers"
a6<-"this nation has never seen by people who waited three hours and four"
a7<-"hours many for the first time in their lives because they believed that"
a8<-"this time must be different, that their voices could be that difference "
speech<-paste(a1,a2,a3,a4,a5,a6,a7,a8,sep=" ")
```

Now using a pie show the distribution of the 10 most used words.

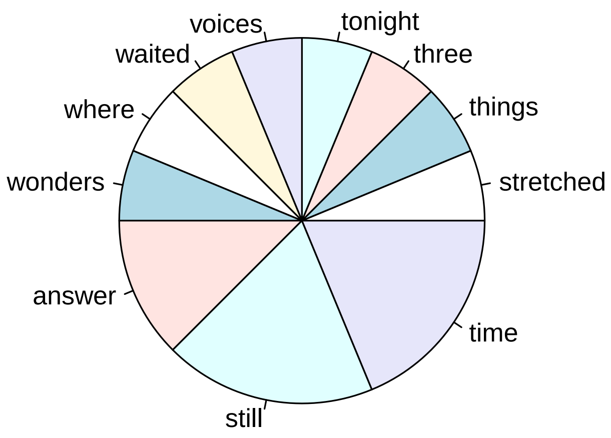
```r
vectorspeech<-unlist(strsplit(speech,split= " "))
pie(sort(table(vectorspeech))[(length(table(vectorspeech))-10):length(table(vectorspeech))])
```



However, if you want to consider the main words only you need to take out the others such that “if”, “is” etc... This is the way to do that

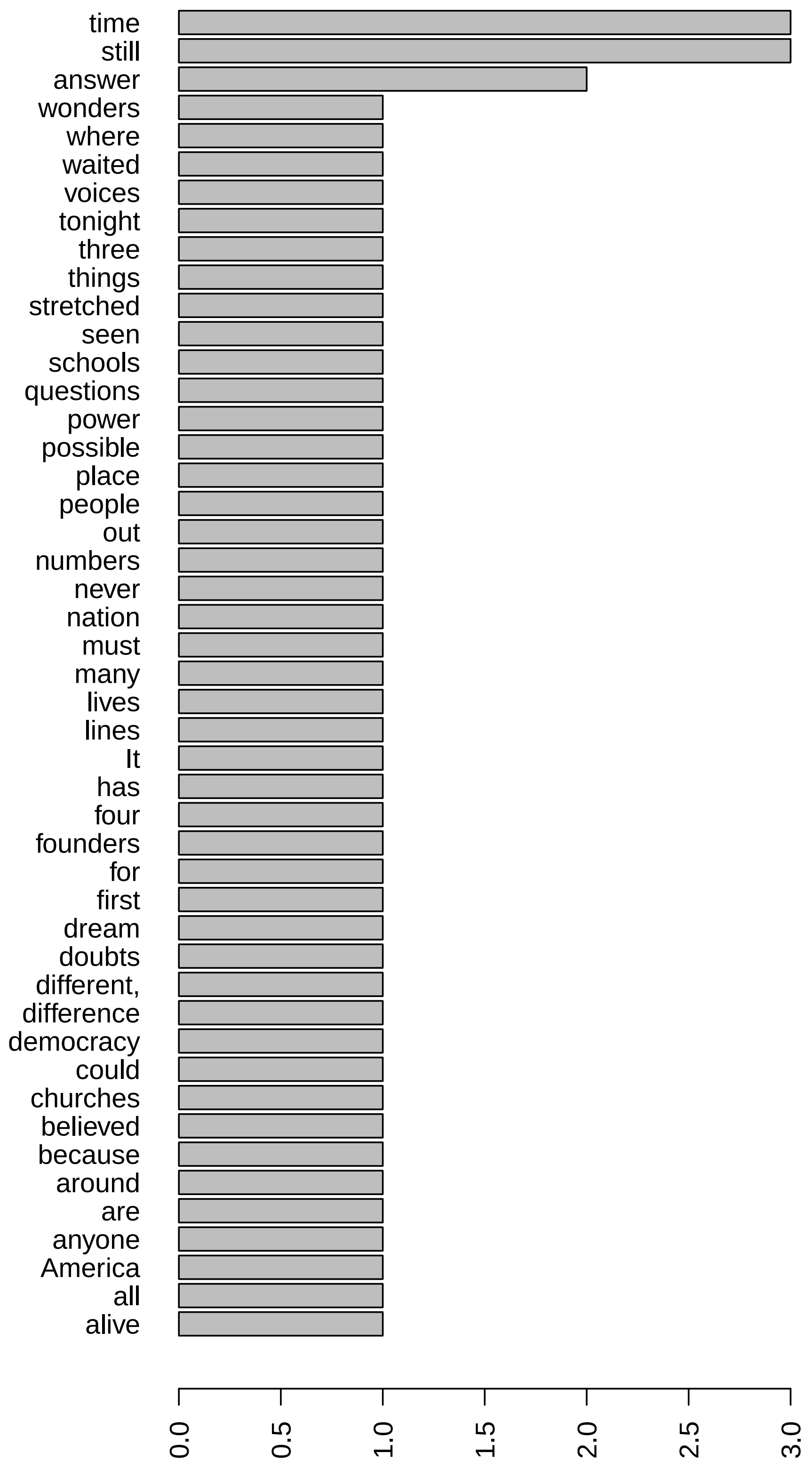
```
tobedeleted<-matrix(0,length(vectorspeech),1);
for(i in 1:length(vectorspeech)){
if(vectorspeech[i]=="is"||
vectorspeech[i]=="by"|| vectorspeech[i]=="I"||
vectorspeech[i]=="a"|| vectorspeech[i]=="if"||
vectorspeech[i]=="If"|| vectorspeech[i]=="in"||
vectorspeech[i]=="the"|| vectorspeech[i]=="that"||
vectorspeech[i]=="of"|| vectorspeech[i]=="our"||
vectorspeech[i]=="be"|| vectorspeech[i]=="there"||
vectorspeech[i]=="this"|| vectorspeech[i]=="their"||
vectorspeech[i]=="and"|| vectorspeech[i]=="they"||
vectorspeech[i]=="your"|| vectorspeech[i]=="hours"||
vectorspeech[i]=="told"|| vectorspeech[i]=="who")
{tobedeleted[i]<- -i}}

vectorspeech<-vectorspeech[c(tobedeleted)]
pie(sort(table(vectorspeech))[(length(table(vectorspeech))-10):length(table(vectorspeech))])
```



The barplot below may be nicer to show:

```r
par(mar=c(5,10,1,1));
barplot(sort(table(vectorspeech)),horiz = TRUE,las=2)
```

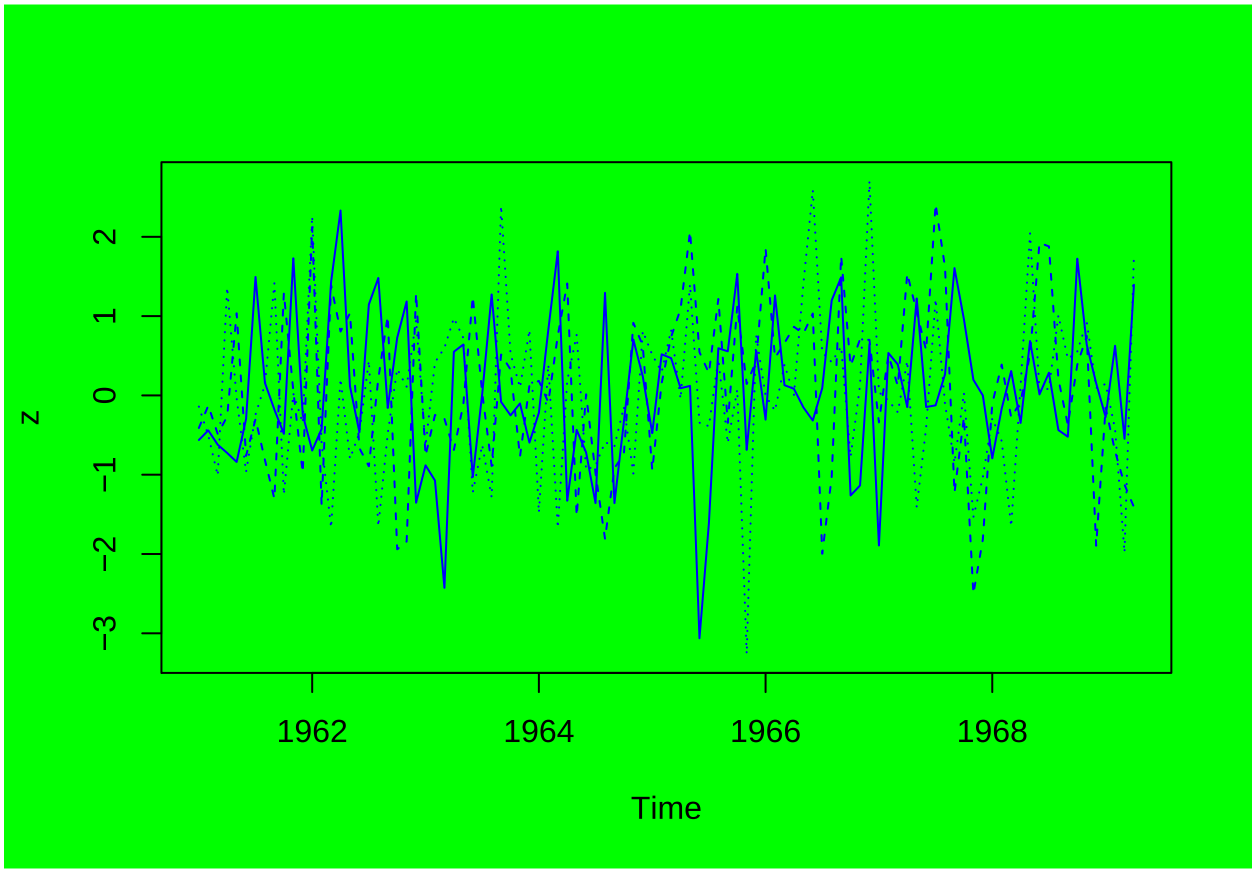


time
still
answer
wonders
where
waited
voices
tonight
three
things
stretched
seen
schools
questions
power
possible
place
people
out
numbers
never
nation
must
many
lives
lines
It
has
four
founders
for
first
dream
doubts
different,
difference
democracy
could
churches
believed
because
around
are
anyone
America
all
alive
0.0
0.5
1.0
1.5
2.0
2.5
3.0

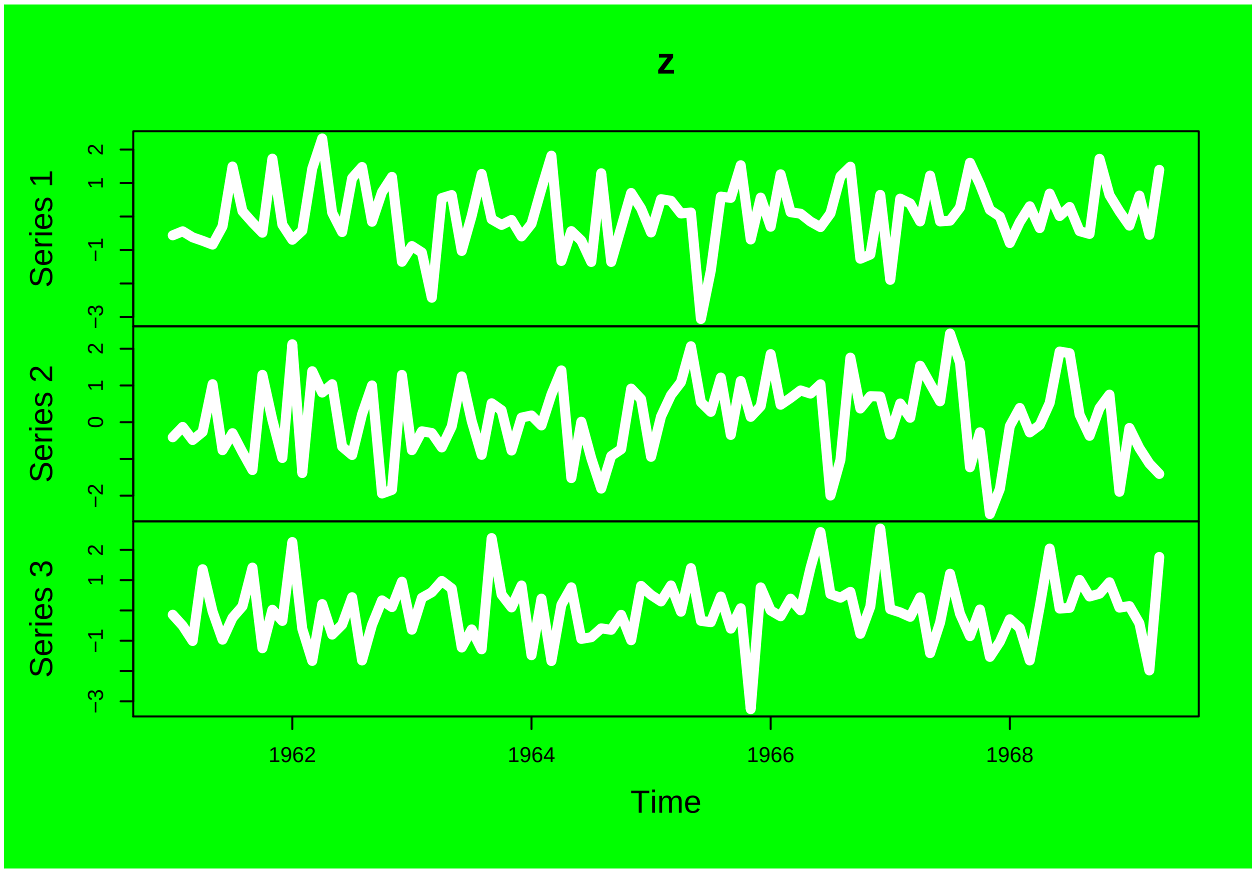
If you want to do more you can download the speeches from this page http://obamaspeeches.com/ and have fun!

Time series should be declared to R as below using ts. Remember to put the start and the frequency (yearly, monthly, quarterly, etc. . .)

```
z <- ts(matrix(rnorm(300), 100, 3), start = c(1961, 1), frequency = 12)
par(bg = "green")
plot(z, plot.type = "single", lty = 1:3, col="blue")
```

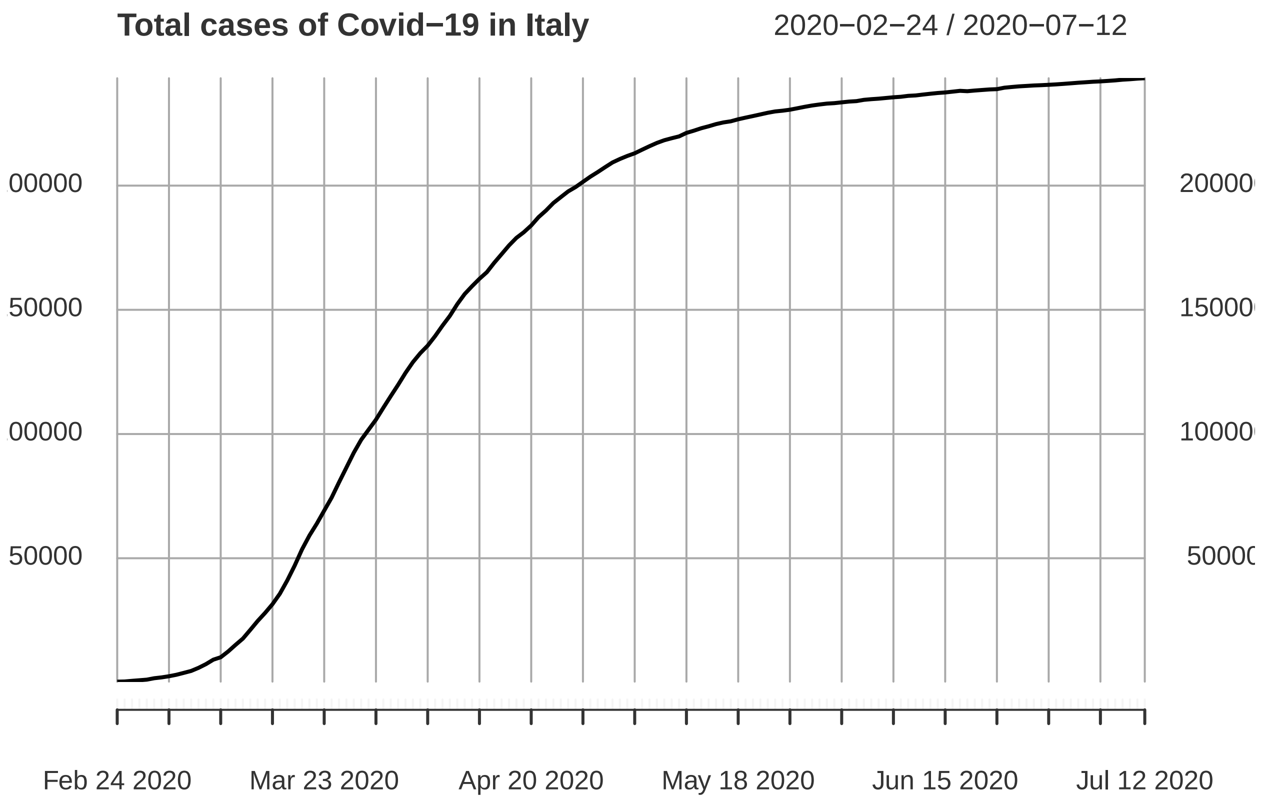


```
plot(z,col="white",lwd=5)
```



Below the graphic of the Covid new cases observed in Italy as shown above but in a smarter format.

```
library(XML)
library("xts")
theURL<-'http://www.pangoo.it/coronavirus/?t=country&r=ITA&data=y#table'
thedata<-readHTMLTable(theURL)
CovidItaly<-as.data.frame(thedata[[1]])
Totalcases<-as.numeric(as.matrix(CovidItaly$V8[2:(length(CovidItaly$V8)-1)]))
starting<-as.Date('2020-02-24')
stopping<-starting+length(Totalcases)-1
dt<-seq(starting,stopping,by = 1)
plot(xts(ts(Totalcases),dt),main="Total cases of Covid-19 in Italy")
```



Exercise Human Resources data

The following script generate data relative to the information of 200 workers working in a firm. Please run the following script:

```
n<-200
sexmat<-matrix(round(runif(n,0,1),0),n,1)
sex<-factor(sexmat,labels=c("Female","Male"))
edumat<-c();wage<-c();jobmat<-c();age<-c()
for(i in 1:n){
if(sexmat[i]==1){
edumat[i]<-round(runif(1,2,3.6),0);
jobmat[i]<-round(runif(1,1.2,4),0);
wage[i]<-2.3+.5*rnorm(1);
age[i]<-round(40+10*rnorm(1),0)
}
else{
edumat[i]<-round(runif(1,1,3.6),0);
jobmat[i]<-round(runif(1,1,2.8),0);
wage[i]<-2+.3*rnorm(1);
age[i]<-round(40+3.5*rnorm(1),0)
}
}
```

```r
edu<-factor(edumat,labels=c("high school","undergrad","postgrad","PhD"))
job<-factor(jobmat,labels=c("intern","admin","profession","manager"))
hrdata<-cbind(wage,age,edu,job,sex);colnames(hrdata)<-c("wage","age","edu","job","sex")
```

Note that the data are stored in the matrix hrdata whose columns have names. Now using the information provided in this class, visualize the data (the best way possible) in order to show what are the main social features of the workers in this firm.

Create a vector of data and plot it using pie

Create a vector of data and plot it using histogram

Create a matrix of data and plot it using boxplot

Create a factor and two other variables and plot the three in a single plot.

What have you learnt during this class

plot(y,main="This is the main title of the plot",xlab = "This is my x label",ylab = "This is my y label",col="blue")

par(mfrow=c(3,2)) #This put six graphics in 3 rows and 2 cols

colnames(matrix)<-c("name1","name2") #This give names to columns of a matrix

rownames(matrix)<-c("name1","name2") #This give names to rows of a matrix

matplot(mymatrix) #this plots the elements of a matrix

plot(something,pch = 1, cex=2, col="black", bg="green", lwd=1) #those are options regarding:

pch=the type of points, cex=size of points, col=color, bg=background, tickness of a line

hist(varuabke,col="yellow",breaks = 10) #plot an histograms with a color and a specific number of bars.

barplot() #similar to hist but you can also plot a table

If you want to plot a matrix with names –> barplot(matrix, beside=TRUE, col=rainbow(5))

boxplot(y) provide the boxplot of the variable y. If you have a matrix it show as many boxplot as many cols.

horizontal=TRUE #show the x's in the y's and the y's in the x's

plot(dfx, dfy,col=df$afactor,pch = 16) #This plot x & y using as colors the factors

legend(add position,legend=levels(thefactor)) #this add a legend in the position specified.

pie(data) #provide a pie chart

strsplit #takes a string of words and split in in several bits, each bit is put into " "

time series can be declared in Rstudio using for example this

ts(myseries, start = c(1961, 1), frequency = 12)

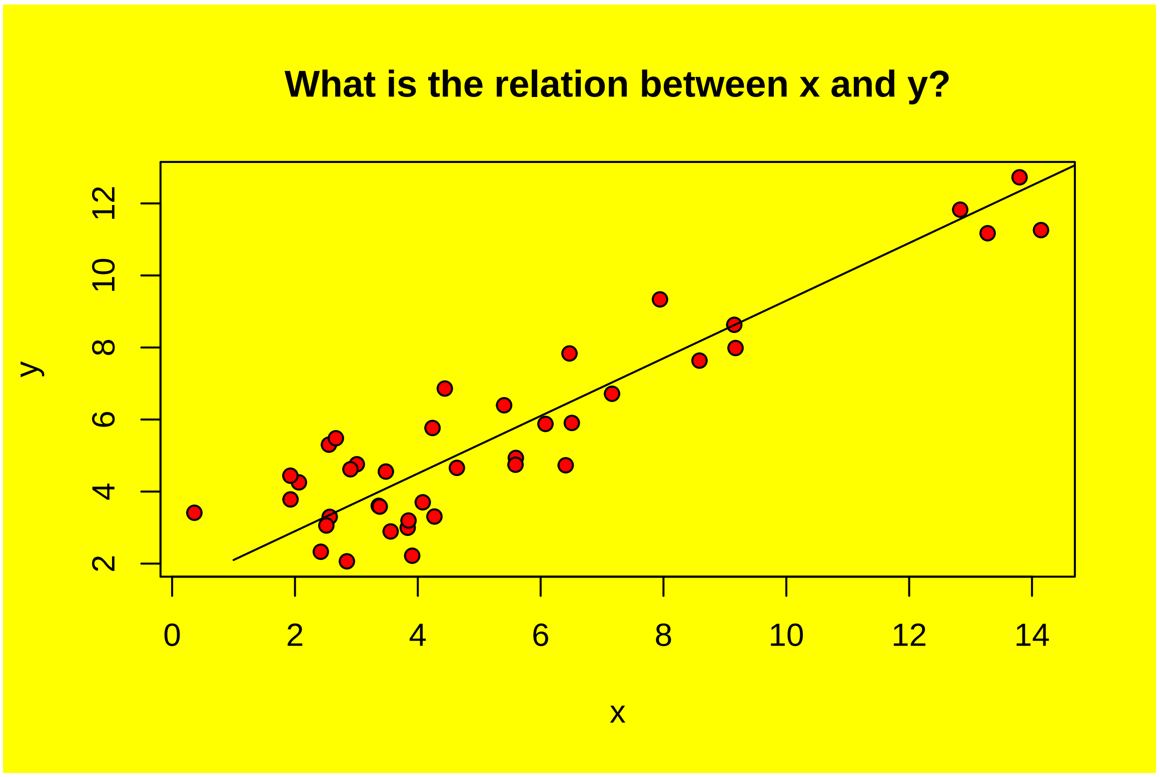
Simple linear regression model

The simple linear regression model can be written as follows:

$y_i = a + b \times x_i + e_i$

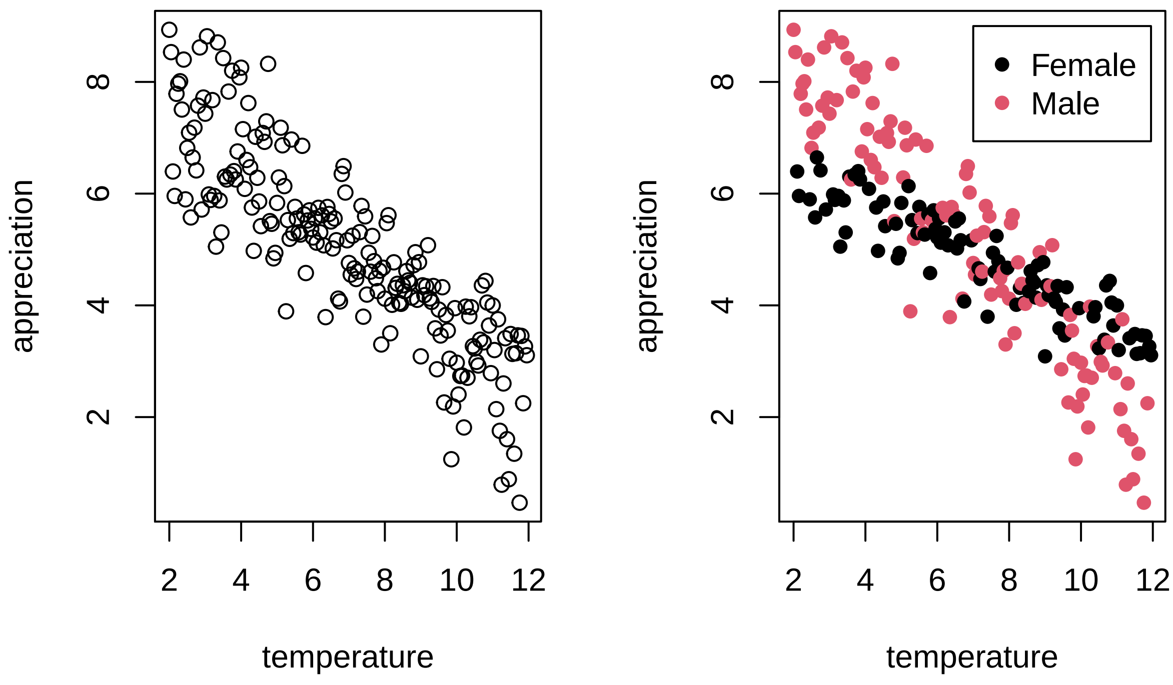
Where y_i is the dependent variable with i=1,2,.,n. x_i is defined as the independent variable while a and b are the regression coefficients. In particular, b measures the impact of x on the y while a is the intercept (constant term) that adds up. Finally, e_i represents the error term containing everything the model does not know about the dependent variable. Note that e_i is also defined as the residuals since we can write: $e_i = y_i - a - b \times x_i$ The parameters of this model are estimated using the so-called Least Squares method (or Ordinary least squares). Lets first generate the model and try to figure out its dynamics using the following code:

```
set.seed(2)
x<-rchisq(40,6)
e<-rnorm(40)
y<-1.3+.8*x+e
pointsline<-1.3+.8*seq(0:20)
par(mfrow=c(1,1),bg="yellow")
plot(x,y, main="What is the relation between x and y?",
     pch = 21, cex=1, col="black", bg="red", lwd=1)
lines(pointsline)
```



For example, we know that the level of appreciation of a beer depends (among other factors) upon the temperature of the beer. Suppose we ask 200 clients (both female and male) to express their appreciation for the beer for each temperature included beteen 2 degres and 12 degrees Celsius. The chunk below is a simulation:

```r
n<-200
set.seed(6)
gender<-round(1.1*runif(n),0)
sex<-factor(gender,labels = c("Female","Male"))
temperature<-2+seq(0,9.99,.05)
ef<-.4*rnorm(n)
em<-.8*rnorm(n)
appreciation<-c()
for(i in 1:n){
  if(gender[i]==0){appreciation[i]<-7-.3*temperature[i]+ef[i]
  } else (appreciation[i]<-10-.7*temperature[i]+em[i])
}
par(mfrow=c(1,2))
plot(temperature,appreciation)
plot(temperature,appreciation,col=sex,pch = 16)
legend(x = 7, y = 9, legend = levels(sex), col = c(1:2), pch = 16)
```



Her we note that there is a negative relation between the temperature and the appreciation of clients in the supermarket. However, it seems that female clients appreciate the beer differently compared to male, when the temperature change. Can you sketch these differences?

We can evaluated these differences by running a linear regression considering the variable appreciation as y and the variable temperature as x (note that here temperature is a control variable that affects the appreciation of the beer). This can be done as follows:

```r
OurRegression=lm(appreciation~temperature)
#If you want to see the results of the regression, in particular:
#the R^2 , the significance of the parameters etc. you need to type this:
summary(lm(appreciation~temperature))   #or   summary(OurRegression)
```

```
##
## Call:
## lm(formula = appreciation ~ temperature)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.27617 -0.50547  0.05136  0.58936  2.15574
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.63210    0.15896   54.30   <2e-16 ***
## temperature -0.51883    0.02106  -24.64   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8597 on 198 degrees of freedom
## Multiple R-squared:  0.754,  Adjusted R-squared:  0.7528
## F-statistic:   607 on 1 and 198 DF,  p-value: < 2.2e-16
```

One can see, indeed, that the level of appreciation decreses toghether with the raising temperature. In addition, each additional degree impact on reducing about 0.5 of the appreciation of the client.

Note that the R^2 is quite high so the temperature seems to be an important factor explaining the dynamics of y.

In addition, both constant and the slope coefficients are statistically significant since we have 3 stars $***$ on the right hand side of the regression coefficients. One has that No star means the parameter is not significantly different from zero (i.e. no impact); 1 star the parameter is moderately significant (at 10% level); 2 stars the parameter is quite significant (at 5% level); 3 stars the parameter is strongly significant (at 1% level).

Now suppose we want to evaluate the beer appreciation by gender. This is because we wish to know if there is the temperature impacts differently in the appreciation of female and that of male. This is the way to run it:

```r
summary(lm(appreciation[gender==0]~temperature[gender==0]))
```

```
##
## Call:
## lm(formula = appreciation[gender == 0] ~ temperature[gender ==
##     0])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.14028 -0.21691  0.01335  0.27434  0.76130
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               6.98410    0.11367   61.44   <2e-16 ***
## temperature[gender == 0] -0.30626    0.01479  -20.71   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4038 on 91 degrees of freedom
## Multiple R-squared:  0.8249, Adjusted R-squared:  0.823
## F-statistic: 428.8 on 1 and 91 DF,  p-value: < 2.2e-16
```

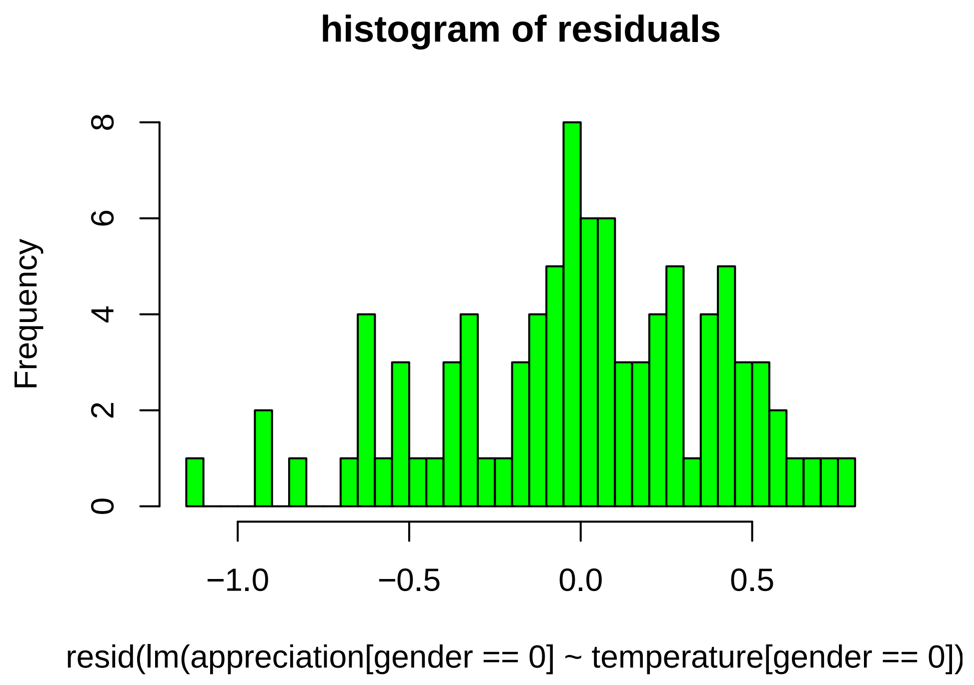
```
summary(lm(appreciation[gender==1]~temperature[gender==1]))
```

```
##
## Call:
## lm(formula = appreciation[gender == 1] ~ temperature[gender ==
##     1])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.39333 -0.51369  0.01213  0.58082  1.69305
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                9.89662    0.19866   49.82   <2e-16 ***
## temperature[gender == 1] -0.68763    0.02675  -25.71   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8096 on 105 degrees of freedom
## Multiple R-squared:  0.8629, Adjusted R-squared:  0.8616
## F-statistic: 660.9 on 1 and 105 DF,  p-value: < 2.2e-16
```

We can see that temperature impacts more on male appreciation compared and less of female. The appreciation of male (say between 10 and 1) is more dispersed than female that tend to be less volatile since the range is between 7 and 3. Thefore, male are more concerned than female regarding the temperature of their beer!

Note that, among the validation criteria of the regression analysis, beside the R^2 and the significance of the parameters (t-test) there is also the normality of Residuals. Firstly, note that if you want to extract the errors or the residuals of your regression you have to type:

```
errors= resid(lm(appreciation[gender==0]~temperature[gender==0]))
#Clearly, you can see the distribution using the histogram:
hist(resid(lm(appreciation[gender==0]~temperature[gender==0])),
     main="histogram of residuals",breaks = 30, col="green")
```



If you want to test the normality of residuals, you have to use the Shaphiro test as follows:

```
shapiro.test(resid(lm(appreciation[gender==0]~temperature[gender==0])))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(lm(appreciation[gender == 0] ~ temperature[gender == 0]))
## W = 0.97712, p-value = 0.1018
```

The null hypothesis is Normality so:

If the p-value is smaller than 5% you reject the Normality hypothesis.

If the p-value is bigger than 5% you accept the Null-hypothesis. The residuals are normal.

Linear regression analysis can use more than one variables (more than one X). Suppose in the previous example we also record the age of the client, using this simulation:

```
n<-200
set.seed(6)
gender<-round(1.1*runif(n),0)
sex<-factor(gender,labels = c("Female","Male"))
temperature<-2+seq(0,9.99,.05)
age<-22+rchisq(n,5)
ef<-.4*rnorm(n)
em<-.8*rnorm(n)
appreciation<-c()
```

```
for(i in 1:n){
  if(gender[i]==0){appreciation[i]<-7-.3*temperature[i]+.1*age[i]+ef[i]
  } else (appreciation[i]<-8-.7*temperature[i]+.02*age[i]+em[i])
}
```

Now if you want to run a regression using two X you can type this:

```
summary(lm(appreciation[gender==0]~temperature[gender==0]+age[gender==0]))
```

```
##
## Call:
## lm(formula = appreciation[gender == 0] ~ temperature[gender ==
##     0] + age[gender == 0])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.70844 -0.29118 -0.03578  0.28567  1.06494
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               6.36176    0.40820  15.585  < 2e-16 ***
## temperature[gender == 0] -0.27761    0.01424 -19.498  < 2e-16 ***
## age[gender == 0]          0.11783    0.01454   8.102 2.51e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3877 on 90 degrees of freedom
## Multiple R-squared:  0.8398, Adjusted R-squared:  0.8362
## F-statistic: 235.9 on 2 and 90 DF,  p-value: < 2.2e-16
```

This adds the variables age and the summary results change accordingly.

Exercise Restaurant

The following code generate a dataset called *Resto*

```
n<-1000
#day<-factor(round(runif(n,1,4),0),labels = c("Mon-Thu","Fri","Sat","Sun"))
CreditCard<-factor(round(runif(n,0,1),0),labels = c("Card","Cash"))
spending<-19+rchisq(n,8)
day<-sex<-timing<-age<-c();
for(i in 1:n){
  if(spending[i]<28){
    sex[i]<-round(runif(1),0)
    timing[i]<-40+rchisq(1,8)
    age[i]<-round(25+rchisq(1,3),0)
    day[i]<-1+3*round(runif(1),0)
    } else {
      sex[i]<-1;
      timing[i]<-90+rchisq(1,10);
      age[i]<-round(45+rchisq(1,6),0)
      day[i]<-round(runif(1,2,4),0)
}
}
day<-factor(day,labels = c("Mon-Thu","Fri","Sat","Sun"))
Resto<-as.data.frame(cbind(spending,age,day,sex,CreditCard,timing))
```

The data frame *Resto* contains the following variables

Variable name	Contents
sex	0 if the worker is female, 1 if the worker is male
Spending	The ammount of spending in Euro of the client
Timing	The number of minutes spent in the Restaurant
Creditcard	Factor indicating if the client pay with “Cash” or “Card”
day	Factor indicating if the client visited on “Mon-Thu”, “Fri”, “Sat” or “Sun”
Age	The age of the client

Using all the graphical and analytical tools you have learnt in the last classes can you make an analysis of the business dynamics in the Restaurant?

What have you learnt during this class?

1) Linear regression analysis using R and RStudio.

2) lm(y~x) #This run a simple linear regression, y function of x and an intercept

3) Summary(lm(y~x)) #providing with more results about the regression such as R^2 significance test of parameters residuals descriptive stats etc.

4) resid() #this provide the errors (residuals) of the regression.

5) Shapiro.test(x) #this is a test of normality that allow testing if the values in x are normally distributed.

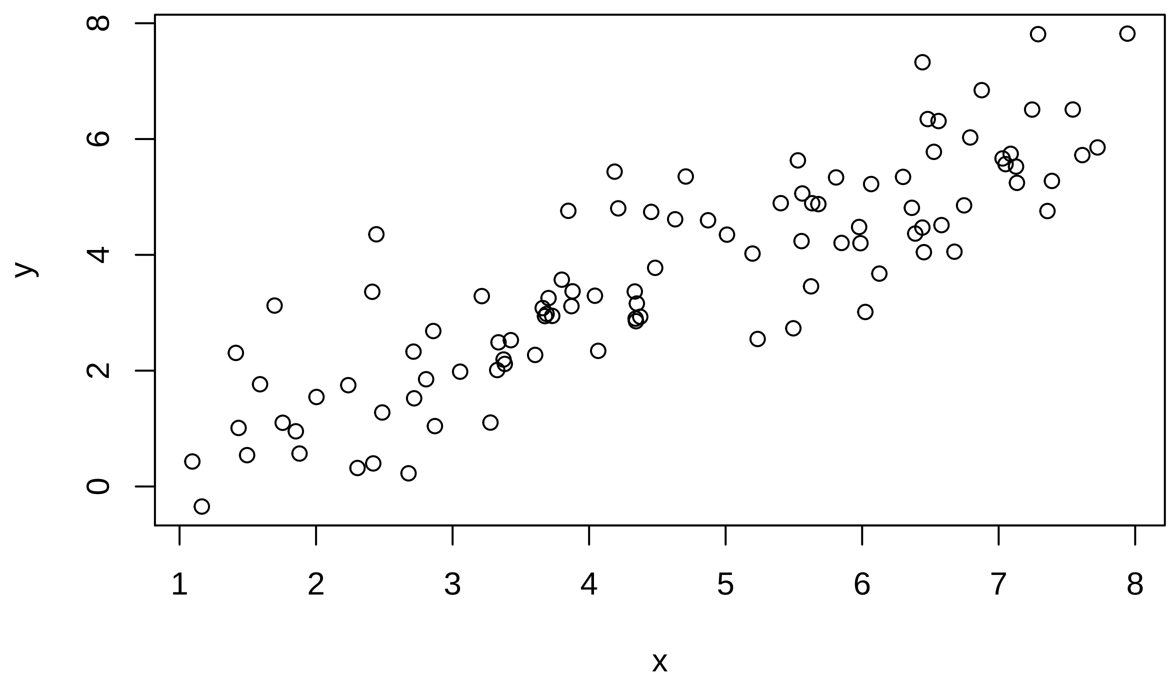
6) Remember that when you run a regression analysis, in order to consider you model you need to check for three validation criteria: 1) the R2 should be bigger than 60-65%. 2) the parameters should be significantly different from zero 3) the residuals are supposed to be normally distributed.

7) lm(y~x+z) #This run a multiple regression y as function of x and z with an intercept

Optim-ization with R

Suppose we observe the following variables, i.e. x and y:

```
set.seed(1)
n<-100
x<-runif(n,1,8)
y<-1*rnorm(n)+.8*x
plot(x,y)
```



Now suppose that we believe that there is a relation between x and y. For example we believe that the following relation holds $y = \beta x + \epsilon$. In other words, we believe that y is function of x and when x changes y changes according to the law expressed in the previous expression.

The following chunk makes the trick:

```
e<-c()
myfu <- function(beta){b<-beta[1];e<-y-b*x;  sum(e^2)};
myrelation<-optim(c(1),myfu,method = "Brent", lower = 0, upper = 3)
myrelation
```

```
## $par
## [1] 0.8029634
##
## $value
## [1] 87.4888
```

```
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Note that we have a function (myfu) whose last element is $sum(e^2)$. Moreover we want to minimize the function by finding a specific value (this is a 1 dimensional optimization). There are several options to optimize a function in R. the *optim* represents a widely used tool. It works like this:

(1) The first part is the vector (in this case a scalar) with the parameter(s) to that optim evaluate in order to minimize the function.

(2) The second part is the function

(3) The third part is the method. Note that Brent works for unidimensional optimization. As we will discuss later, when you have two or more than to parameters (multidimensional optimization) you can either avoid specifying the method or specify a specific method (this option we do not consider for ease of comprehension)

An alternative code, which is more involved since it run a for loop before providing the element we want to minimize, is the following:

```
e<-c()
myfu <- function(beta){b<-beta[1]
  for (i in 1:(n)) {
    e[i]<-y[i]-b*x[i]
        }
  sum(e^2)}

myrelation<-optim(c(1),myfu,method = "Brent", lower = 0, upper = 3)
myrelation
```

```
## $par
## [1] 0.8029634
##
## $value
## [1] 87.4888
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

As you can see the results are the same as before! You might want to change what you want to minimize. For example, suppose you want to minimize the sum of the absolute errors. Then the following code makes the trick.

```
e<-c()
myfu <- function(beta){b<-beta[1];e<-y-b*x;  sum(abs(e))};
myrelation<-optim(c(1),myfu,method = "Brent", lower = 0, upper = 3)
myrelation
```

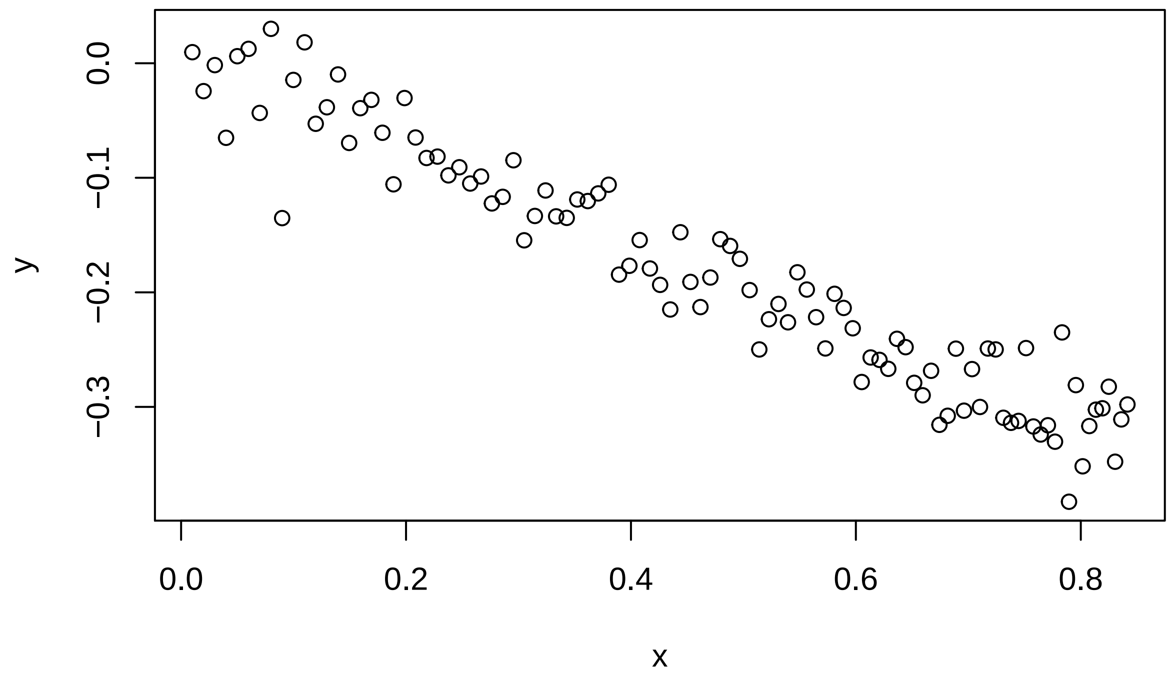
```
## $par
## [1] 0.78245
##
## $value
## [1] 73.71176
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Note however that the results do not correspond as above.

Exercise (unidimensional)

Consider the following code:

```
set.seed(1224)
x<-sin(seq(0.01,1,by=.01));
b<-runif(1);
y<--b*x+.03*rnorm(length(x));
plot(x,y)
```



Suppose we believe that there exist the following relation $y = \beta x + \epsilon$.

Can you find the value of beta that minimize the sum of squared epsilon?

Can you find the value of beta that minimize the sum of absolute epsilon?

```
e<-c()
myfu <- function(beta){b<-beta[1];e<-y-b*x;  sum(abs(e))};
myrelation<-optim(c(1),myfu,control=list(fnscale=1),method = "Brent", lower = -4, upper = 4)
myrelation
```

```
## $par
## [1] -0.3974674
##
## $value
## [1] 2.392919
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
```

```
##
## $message
## NULL
```

Suppose that we want to maximize a function. Consider the function $y = \frac{(\exp -0.5(x-4)^2)}{\sqrt{2\pi}}$ (By the way do you recogize what function is this? If not, try to make a plot of it!). Now suppose we wish to maximize it. Then we should use the option $control = list(fnscale = -1)$ inside optim. For example

```
myfun<-function(g){y<- (exp(-.5*(g-4)^2)/sqrt(2*pi));y}
myrelation<-optim(c(1),myfun,control=list(fnscale=-1),method="Brent",lower=-14,upper =14)
print(myrelation)
```

```
## $par
## [1] 4
##
## $value
## [1] 0.3989423
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

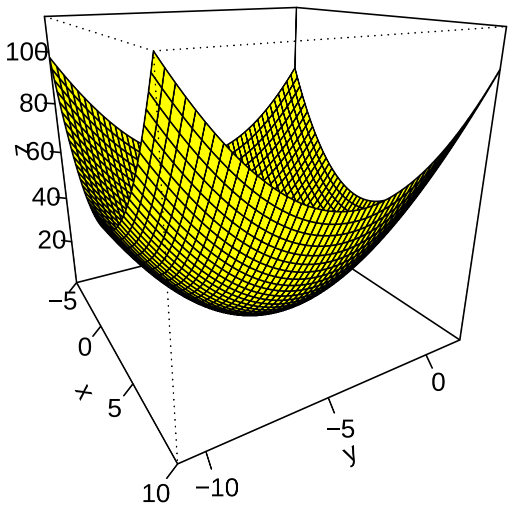
Now suppose we want to minimize a function of two variables. For example, consider the following function $z = (-2+x)^2 + (4+y)^2$ and assume we want to find the values of x and y that minimize z. Then we can still use optim as follows:

```
myfun<-function(myvalues){x<-myvalues[1];y<-myvalues[2];
(x-2)^2+(4+y)^2}
myresults<-optim(c(1,3),myfun)
print(myresults)
```

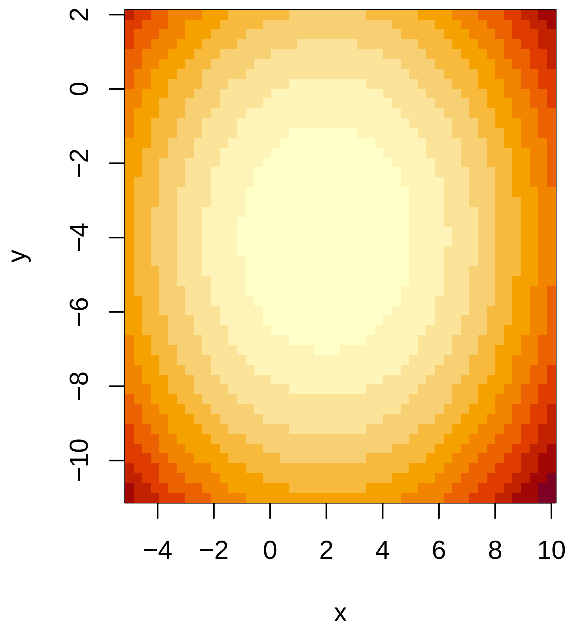
```
## $par
## [1]  1.999499 -4.000202
##
## $value
## [1] 2.915144e-07
##
## $counts
## function gradient
##       67       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

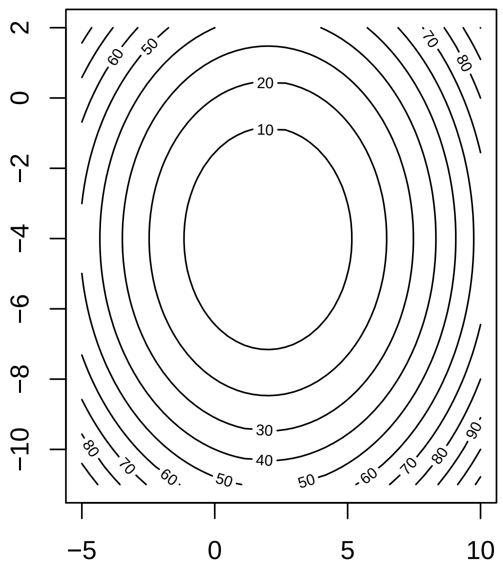
Suppose we want to have some fun by visualizing this z function as above. This is a nice code that allows you to see the beauty & power of R-graphics:

```
x<-seq(-5,10,len=50);
y<-seq(-11,2,len=50);
f<-function(x,y) {(-2+x)^2+(4+y)^2};
z<-outer(x,y,f);
persp(x,y,z,phi=20,theta=60,col="yellow",ticktype="detailed")
```

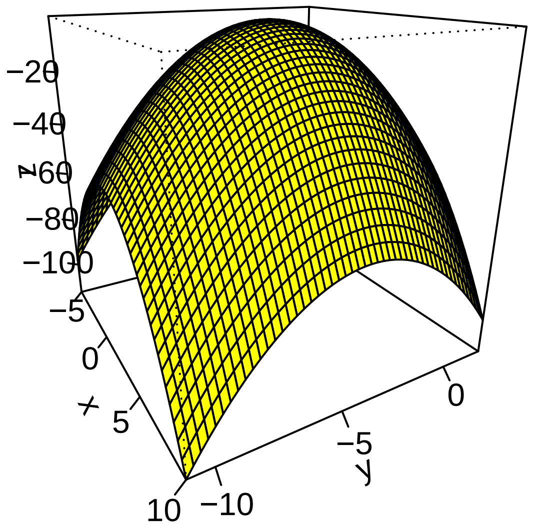


```
par(mfrow=c(1,2));
image(x,y,z)
contour(x,y,z)
```





```
x<-seq(-5,10,len=50);
y<-seq(-11,2,len=50);
f<-function(x,y) {-(-2+x)^2-(4+y)^2};
z<-outer(x,y,f);persp(x,y,z,phi=20,theta=60,col="yellow",ticktype="detailed")
```



Now consider the following function $\exp\left(-0.01\left((x-1)^2+(y-2)^2-0.5(x-3)(y-4)\right)\right)$. Can you find the values of x and y that maximize it? Can you plot the function in 3 Dimension?

Excercise Optim with 2 parameters

Impose set.seed(123)

Generate a vector x (of dimension n=100) using a uniform distribution between 3 and 10. Moreover generate a constant $c = 7$ and a scalar $b = .9$. Moreover generate the following variable $y = c + b * x + rnorm(length(x))$

Now suppose we do not know the existence of this relation but we try to find it using a function that, using the vector $para$, minimizes the sum of squared errors that is the sum of squared $e = y - para[1] - para[2] * x$.

Show the results and make some meaningful comments.

The Normal distribution

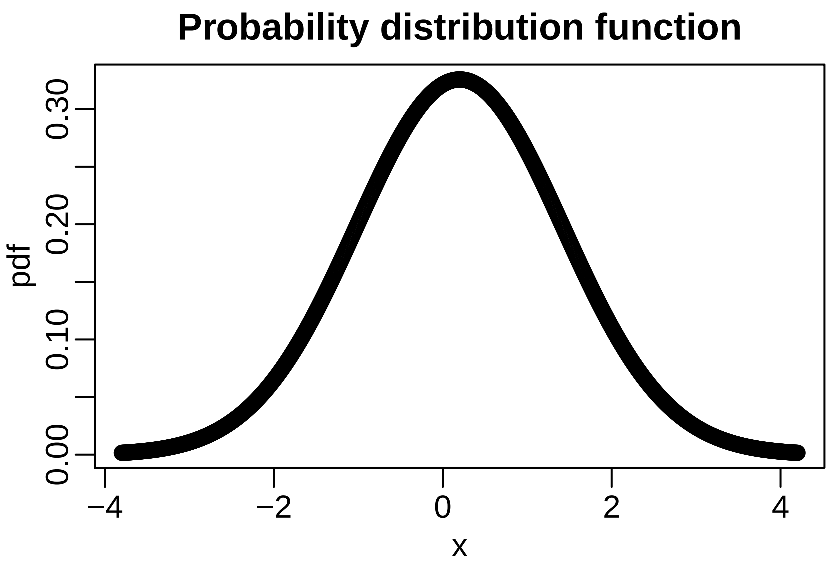
The normal (or Gaussian) probability function represents a crucial distribution in statistics. Most of the statistical inference is based on that distribution. One remarkable example is the so-called Central Limit Theorem, one of the most important result achieved in the last centuries! As a matter of fact, the properties of the (multivariate) Normal distribution represent also the cornerstone of the so-called Kalman filter, a crucial tool widely used in engineering, finance and economics. Therefore, before introducing the Kalman filter and its use for filtering (forecasting) we will first focus on the univariate and bivariate Normal distribution properties. For the sake of simplicity and since we will focus on the filtering of a single variable, We will consider only the bivariate Normal properties. However, similar results also hold for the generic multivariate Normal distribution. This is left for the future. The aim of the following sections is to provide a theoretical, yet practical, approach for understanding the basics of the Kalman filter and its potentialities in data analysis. The reader will face simultaneously both theory and examples codes using R. This might be an interesting approach differing from most of the books in time series analysis. The emphasis here is not to follow the standard practice of teaching time series. On the contrary, the aim is to provide the basic knowledge to understand and simplify the state-space modeling approach by also introducing recently published results (by the author).

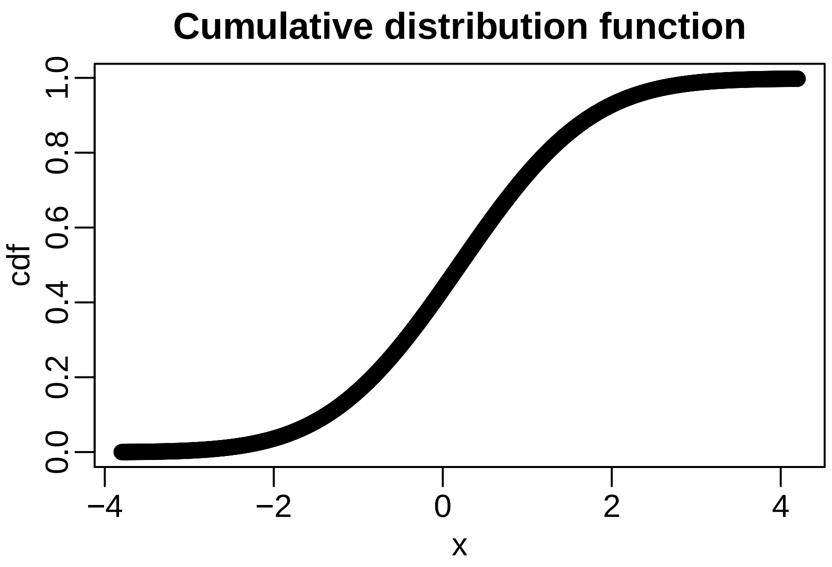
Consider a Normal distributed variable X having mean μ and standard deviation σ. We can then write the so called probability density function (pdf) for the univariate case as follows:

$$Probability(x) = \frac{1}{\sqrt{(2\pi\sigma^2)}} exp(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}]$$

Suppose we want to generete a normal distribution with mean 0.2 and variance 1.5 with R. This simple code does the job:

```
x<-seq(-3.8,4.2,len=800);
mu<-.2
sigma<-sqrt(1.5)
pdf<-c();
cdf<-c();
for(i in 1:length(x)){
  pdf[i]<-(1/sqrt(2*pi*sigma^2))*exp(-.5*((x[i]-mu)^2/sigma^2));
  cdf[i]=round(sum(pdf[1:i])/100,4)};
par(mfrow=c(2,1),mar=c(2.5,4,2,2), mgp=c(1.5,.5,0))
plot(x,pdf, xlab="x",main="Probability distribution function")
plot(x,cdf, xlab="x",main="Cumulative distribution function")
```





Where the last graph is the plot of the cumulative distribution function.

Any normal distribution can be “standardized" as follows

$$z = \frac{x - \mu}{\sigma}$$

This has the same bell shape as x but with mean 0 and variance 1. Use the above code to generate and plot the pdf and the cdf of the standard normal distribution.

Bivariate Normal distribution

Suppose that two variables, say x and y, are jointly normal distributed with mean respectively μ_x, μ_y and variance respectively σ_x^2 and σ_y^2. Moreover suppose that the covariance between the two variables is σ_{xy}. This can be represented as follows:

$$\begin{bmatrix} x \\ \\ y \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_x \\ \\ \mu_y \end{bmatrix}; \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix}\right)$$

Where the so called variance/covariance matrix is:

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix}$$

For practical reason we need to derive the inverse of this matrix, say

$$\Sigma^{-1}$$

such that

$$\Sigma\Sigma^{-1} = I$$

. One can see that this matrix is:

$$\Sigma^{-1} = \begin{bmatrix} \frac{\sigma_y^2}{\sigma_x^2\sigma_y^2 - \sigma_{xy}^2} & \frac{-\sigma_{xy}}{\sigma_x^2\sigma_y^2 - \sigma_{xy}^2} \\ \\ \frac{-\sigma_{xy}}{\sigma_x^2\sigma_y^2 - \sigma_{xy}^2} & \frac{\sigma_x^2}{\sigma_x^2\sigma_y^2 - \sigma_{xy}^2} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}} & \frac{\frac{-\sigma_{xy}}{\sigma_y^2}}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}} \\ \\ \frac{\frac{-\sigma_{xy}}{\sigma_y^2}}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}} & \frac{\frac{\sigma_x^2}{\sigma_y^2}}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}} \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 \\ \\ -\frac{\sigma_{xy}}{\sigma_y^2} & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}} & 0 \\ \\ 0 & \frac{1}{\sigma_y^2} \end{bmatrix} \begin{bmatrix} 1 & -\frac{\sigma_{xy}}{\sigma_y^2} \\ \\ 0 & 1 \end{bmatrix}$$

Let's verify with R if this is true:

```r
#Consider the following matrix called A
A<-matrix(.7,2,2);A[1,1]<-1;A[2,2]<-1.4;A
```

```
##      [,1] [,2]
## [1,]  1.0  0.7
## [2,]  0.7  1.4
```

```r
#Now performing the previous calculation the inverse of A (defined here as InvA) is:
InvA<-matrix(1/(A[1,1]*A[2,2]-A[1,2]^2),2,2);InvA[1,1]<-InvA[1,1]*A[2,2];
InvA[2,2]<-InvA[2,2]*A[1,1];InvA[2,1]<-InvA[1,2]<-InvA[2,1]*-A[1,2]
InvA
```

```
##            [,1]       [,2]
## [1,]  1.5384615 -0.7692308
## [2,] -0.7692308  1.0989011
```

```r
#Now if we invert the matrix A using the command solve we have:
solve(A)
```

```
##            [,1]       [,2]
## [1,]  1.5384615 -0.7692308
## [2,] -0.7692308  1.0989011
```

```
#So we checked that the result is true!
B<-diag(1,2,2);B[2,1]<- -A[1,2]/A[2,2];C<-diag(1,2,2);
C[1,1]<-1/(A[1,1]-A[1,2]^2/A[2,2]);C[2,2]<-1/A[2,2];
B%*%C%*%t(B)
```

```
##            [,1]       [,2]
## [1,]  1.5384615 -0.7692308
## [2,] -0.7692308  1.0989011
```

```
#Finally we check:
A%*%InvA
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

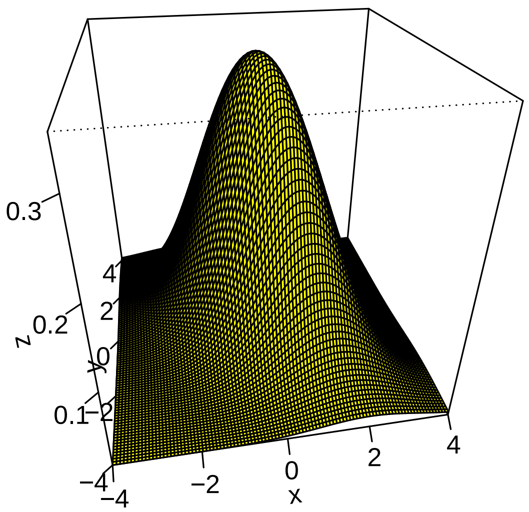
```
#Which is the identity matrix confirming the result
```

Now the pdf of the bivariate normal can be written as:

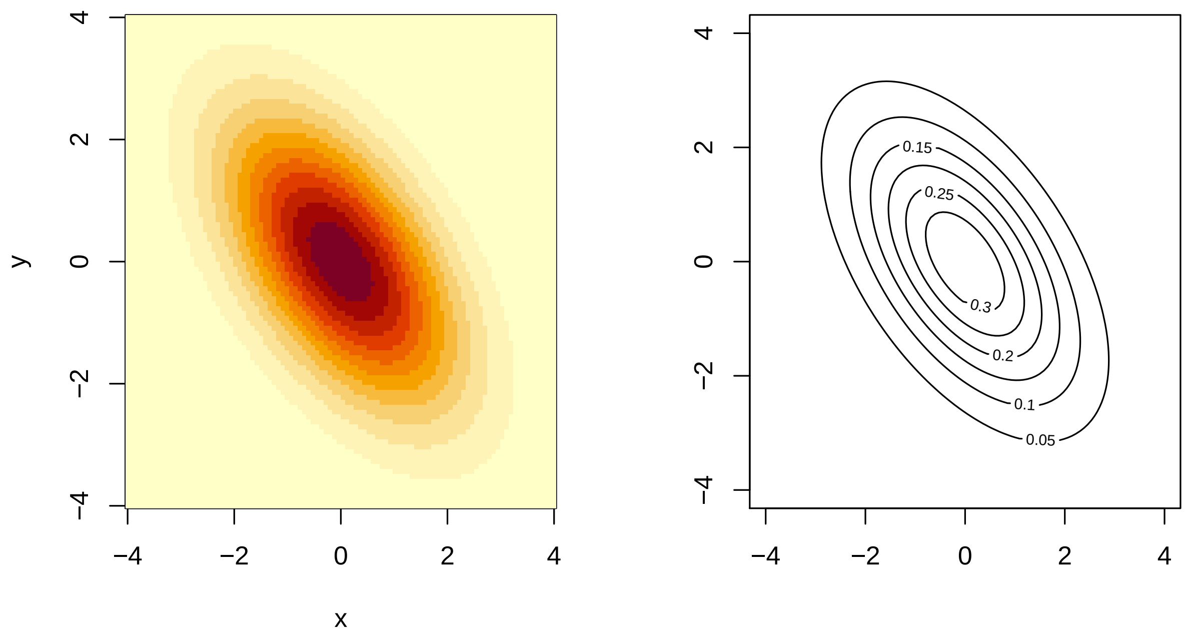
$$p(x,y) = \frac{1}{2\pi|\Sigma|^{\frac{1}{2}}} exp(-0.5 \begin{bmatrix} x-\mu_x & y-\mu_y \end{bmatrix} \Sigma^{-1} \begin{bmatrix} x-\mu_x \\ y-\mu_y \end{bmatrix})$$

Where $|M|$ is the determinant of a matrix M. Here we have $|\Sigma| = \sigma_x^2\sigma_y^2 - \sigma_{xy}^2$.

```
mx<-0
my<-0
varx<-.5
vary<-.6
covxy<- -.3
Sigma<-matrix(NA,2,2);Sigma[1,1]<-varx;Sigma[2,2]<-vary;
Sigma[1,2]<-Sigma[2,1]<-covxy;
x<-seq(-4,4,len=100);
y<-seq(-4,4,len=100);
#f<-function(x,y) {(1/(2*pi*det(Sigma)))*
#exp(-.5*((c(x-mx,y-my))%*%solve(Sigma)%*%c(x-mx,y-my)))}
f<-function(x,y) (1/(2*pi*det(Sigma)^.5))*
  exp(-.5*((vary*(x-mx)^2+(y-my)*(-2*(x-mx)*covxy+varx*(y-my)))/(varx*vary-2*covxy)))
z<-outer(x,y,f);
persp(x,y,z,phi=30,theta=-10,col="yellow",ticktype="detailed")
```



```
par(mfrow=c(1,2));
image(x,y,z)
contour(x,y,z)
```



Using the results as above one obtains:

$$p(x,y) = \frac{1}{2\pi\left(\sigma_x^2\sigma_y^2 - \sigma_{xy}^2\right)^{\frac{1}{2}}} \times$$
$$\times exp(-0.5 \begin{bmatrix} \left(x-\mu_x - \frac{\sigma_{xy}}{\sigma_y^2}(y-\mu_y)\right) & y-\mu_y \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}} & 0 \\ 0 & \frac{1}{\sigma_y^2} \end{bmatrix} \begin{bmatrix} \left(x-\mu_x - \frac{\sigma_{xy}}{\sigma_y^2}(y-\mu_y)\right) \\ y-\mu_y \end{bmatrix})$$

Conditional Normal distribution

Since we discussed about both univariate and bivariate normal distribution, it is now relevant to introduce the conditional distribution. In probability, the definition of conditional distribution is the following:

$$p(x|y) = \frac{p(x,y)}{p(y)}$$

That is: the probability of x given a specific y is the ratio between the joint (bivariate) distribution and the univariate distribution of y.

Now define:

$$\left(x-\mu_x - \frac{\sigma_{xy}}{\sigma_y^2}(y-\mu_y)\right) = x - \mu_{x|y}$$

Then we have:

$$p(x,y) = \frac{1}{2\pi\left(\sigma_x^2\sigma_y^2 - \sigma_{xy}^2\right)^{\frac{1}{2}}} exp(-\frac{1}{2} \times \left[\frac{\left(x-\mu_x - \frac{\sigma_{xy}}{\sigma_y^2}(y-\mu_y)\right)^2}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}} + \frac{(y-\mu_y)^2}{\sigma_y^2}\right])$$

We can now derive the conditional probability of x given y. This is:

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{\frac{1}{2\pi\left(\sigma_x^2\sigma_y^2-\sigma_{xy}^2\right)^{\frac{1}{2}}} exp\left(-\frac{1}{2} \times \left[\frac{\left(x-\mu_x-\frac{\sigma_{xy}}{\sigma_y^2}(y-\mu_y)\right)^2}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}} + \frac{(y-\mu_y)^2}{\sigma_y^2}\right]\right)}{\frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{1}{2}\frac{(y-\mu_y)^2}{\sigma_y^2}\right)})$$

That can be written as:

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{1}{\sqrt{2\pi}\left(\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}\right)^{\frac{1}{2}}} exp\left(-\frac{1}{2} \times \left[\frac{\left(x-\mu_x - \frac{\sigma_{xy}}{\sigma_y^2}(y-\mu_y)\right)^2}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}} + \frac{(y-\mu_y)^2}{\sigma_y^2} - \frac{(y-\mu_y)^2}{\sigma_y^2}\right]\right]$$

This simplifies as:

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{1}{\sqrt{2\pi\left(\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}\right)}} exp\left(-\frac{1}{2} \times \left[\frac{\left(x-\mu_x - \frac{\sigma_{xy}}{\sigma_y^2}(y-\mu_y)\right)^2}{\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}}\right]\right]$$

Which is again a univariate (yet conditional!) normal distribution with mean:

$$\mu_{x|y} = \left(\mu_x + \frac{\sigma_{xy}}{\sigma_y^2}(y - \mu_y)\right)$$

and variance

$$\sigma_x^2 - \frac{\sigma_{xy}^2}{\sigma_y^2}$$

. Note that $\left(\mu_x + \frac{\sigma_{xy}}{\sigma_y^2}(y - \mu_y)\right)$ is the mean of x conditional on the value of y. That is the mean of x imposing the information provided by the variable y. This identity is rather relevant since it is key in the derivation of the Kalman filter as shown below.

State-Space models and the Kalman filter

Consider the following State-Space model:

$$\begin{aligned} y_t &= z\alpha_{t-1} + e_t \\ \alpha_t &= c + w\alpha_{t-1} + u_t \end{aligned} \tag{1}$$

We define α_t as the state variable (the one which is unobserved) and y_t the data that we observe (some people define it the observational equation).

Note that c is a constant and $t = 1, 2, \cdots, n$ is the time such that we assume that we observe n observations. Assuming that e_t and u_t are uncorrelated normal distributed variables such that $e_t \sim N(0, \sigma_e^2)$ and $u_t \sim N(0, \sigma_u^2)$ with $E(e_{t-j}u_{t-i}) = 0$ for any j and i. Since a linear combination of normal (or Gaussian) variables remains normal we have that the vector

$$\begin{bmatrix} y_t \\ \alpha_t \end{bmatrix}$$

follows a bivariate normal distribution. Now defining $E(\alpha_t) = c + wE(\alpha_{t-1})$, we have that the previous vector is Normal with mean

$$\begin{bmatrix} zE(\alpha_{t-1}) \\ c + wE(\alpha_{t-1}) \end{bmatrix}$$

While the matrix/covariance matrix of the vector is

$$\Sigma = \begin{bmatrix} z^2E\left(\alpha_{t-1} - E[\alpha_{t-1}]\right)^2 + \sigma_e^2 & zwE\left(\alpha_{t-1} - E[\alpha_{t-1}]\right)^2 \\ zwE\left(\alpha_{t-1} - E[\alpha_{t-1}]\right)^2 & w^2E\left(\alpha_{t-1} - E[\alpha_{t-1}]\right)^2 + \sigma_u^2 \end{bmatrix}$$

Now imagine that we observe y_t, but we do not observe α_t. This may represents an obstacle since we are not able to know neither $E(\alpha_t)$ nor $E\left(\alpha_t - E[\alpha_t]\right)^2$. However, we can make use of the conditional distribution properties, as above, to derive $E(\alpha_t|y_t)$ and $p_t = Variance(\alpha_t|y_t) = E\left(\alpha_t - E[\alpha_t|y_t]\right)^2$:

Therefore we can now derive the conditional probability of α_t given the information provided by y_t. Now define the information up to time t as follows: $Y_t = [y_1, y_2, \cdots, y_t]$. Moreover, define $E(\alpha_t|Y_t) = a_t$ and $v_t = y_t - zE(\alpha_{t-1}|Y_{t-1}) = z[\alpha_{t-1} - a_{t-1}] + e_t$ such that

$$E(v_t^2) = z^2E[\alpha_{t-1} - E(\alpha_{t-1}|Y_{t-1}) + e_t]^2 = z^2p_{t-1} + E[e_t]^2 = z^2p_{t-1} + \sigma_e^2$$

.

Note also that $\sigma_y^2 = E(v_t^2) = z^2 p_{t-1} + \sigma_e^2$. It then follows that

$$a_t = E(\alpha_t|Y_t) = c + wE(\alpha_{t-1}|Y_{t-1}) + \frac{zwp_{t-1}}{z^2p_{t-1}+\sigma_e^2}\left(y_t - zE(\alpha_{t-1}|Y_{t-1})\right) = c + wa_{t-1} + \frac{zwp_{t-1}}{z^2p_{t-1}+\sigma_e^2}v_t$$

For simplicity, we define $k_t = \frac{zwp_{t-1}}{z^2p_{t-1}+\sigma_e^2}$. In addition, we also have that the conditional variance is:

$$\begin{aligned}
&p_t = E\left[\alpha_t - E(\alpha_t|Y_t)\right]^2 = \\
&E\left[c + w\alpha_{t-1} + u_t - c - wa_{t-1} - k_t v_t\right]^2 = \\
&E\left[w(\alpha_{t-1} - a_{t-1}) + u_t - k_t\left(z[\alpha_{t-1} - a_{t-1}] + e_t\right)\right]^2 = \\
&E\left[(w - k_t z)\left(\alpha_{t-1} - a_{t-1}\right) + u_t - k_t e_t\right]^2 = \\
&\left(w - k_t z\right)^2 E(\alpha_{t-1} - a_{t-1})^2 + \sigma_u^2 + k_t^2\sigma_e^2 = \\
&w^2 p_{t-1} - 2wzk_t p_{t-1} + k_t^2(z^2 p_{t-1} + \sigma_e^2) + \sigma_u^2 = \\
&w^2 p_{t-1} - wzk_t p_{t-1} + \sigma_u^2
\end{aligned}$$

We can now write the following distribution for the vector

$$\begin{bmatrix} y_t \\ \alpha_t \end{bmatrix} |Y_{t-1} \sim N\left(\begin{bmatrix} za_{t-1} \\ c + wa_{t-1} \end{bmatrix}; \begin{bmatrix} z^2p_{t-1}+\sigma_e^2 & wzp_{t-1} \\ wzp_{t-1} & w^2p_{t-1} - \frac{(zwp_{t-1})^2}{z^2p_{t-1}+\sigma_e^2} + \sigma_u^2 \end{bmatrix}\right)$$

We can now collect the main recursions that we have been discussed so far:

$$\begin{aligned}
p_t &= w^2 p_{t-1} - wzk_t p_{t-1} + \sigma_u^2 \\
k_t &= \frac{zwp_{t-1}}{z^2p_{t-1}+\sigma_e^2} \\
a_t = E(\alpha_t|y_t) = c + wa_{t-1} + \frac{zwp_{t-1}}{z^2p_{t-1}+\sigma_e^2}\left(y_t - za_{t-1}\right) &= c + wa_{t-1} + k_t\left(y_t - za_{t-1}\right) \\
v_t &= \left(y_t - za_{t-1}\right)
\end{aligned} \tag{2}$$

These recursions are the well-known Kalman filter!

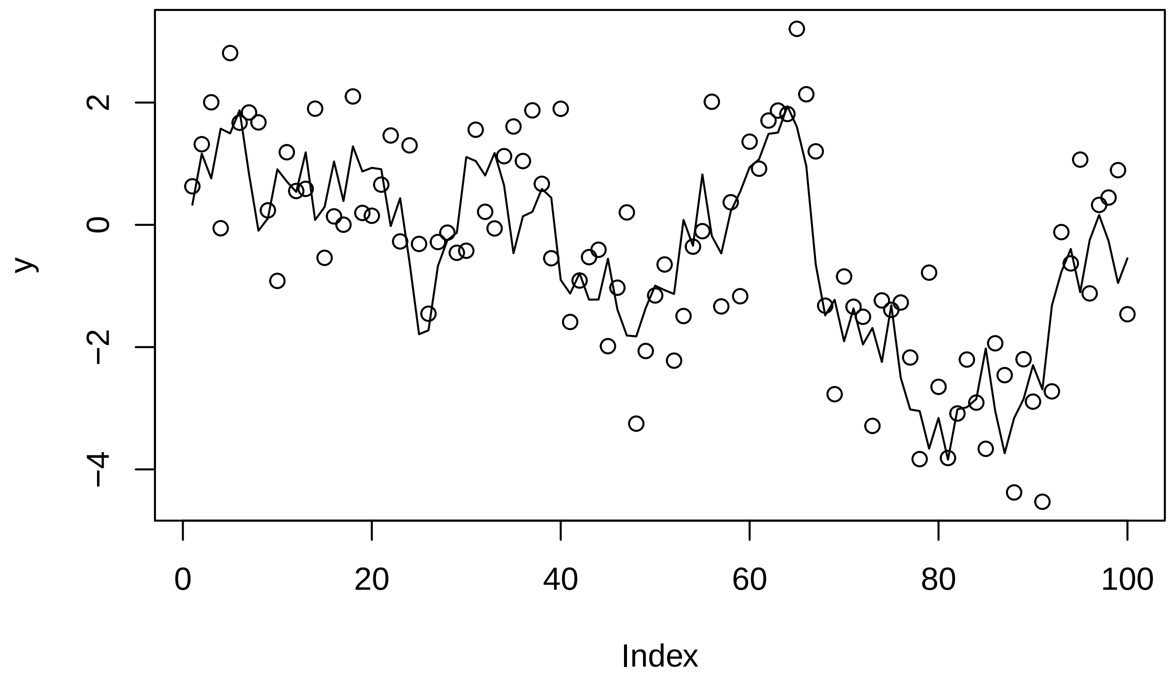
Kalman filter example

Consider the following model:

$$\begin{aligned}
y_t &= \alpha_{t-1} + e_t \\
\alpha_t &= .9\alpha_{t-1} + u_t
\end{aligned} \tag{3}$$

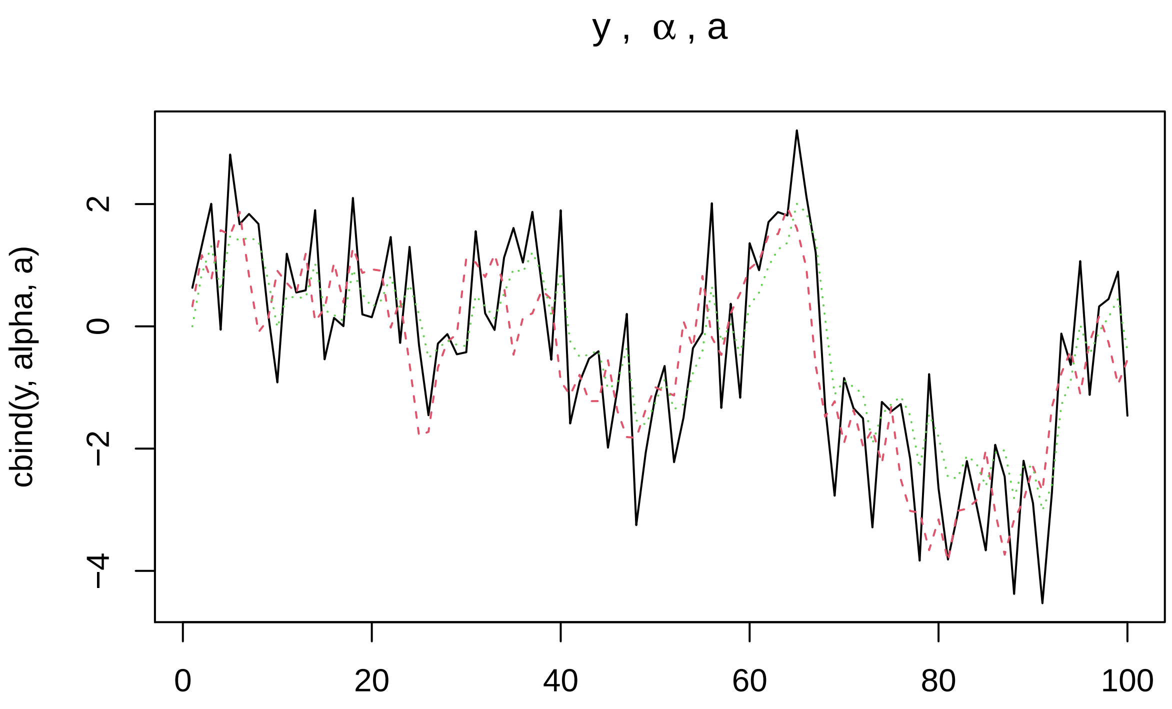
Where $\sigma_e^2 = .8$ and $\sigma_u^2 = .4$. Now we can generate it using the following code:

```r
n<-100
set.seed(1123)
e<-sqrt(.8)*rnorm(n)
u<-sqrt(.4)*rnorm(n)
y<-alpha<-c()
y[1]=e[1];alpha[1]<-u[1]
for(t in 2:n){
  y[t]<-alpha[t-1]+e[t]
  alpha[t]<-.9*alpha[t-1]+u[t]
    }
plot(y,type="p")
  lines(alpha,type="l")
```



We now implement the Kalman filter as in (2). First note that since the state equation α_t is an Autoregressive process of order one we can initialize $a_1 = 0$ and $p_1 = \frac{\sigma_u^2}{1-0.81} = 2.11$ (the last is the variance of α_t). Now, assuming that we know the variances of the noises (this is not the case in empirical situations) we can make use of the following code:

```
n<-100
sigmae<-.8;sigmau<-.4;w<-.9;z<-1
a<-c();p<-c()
a[1]=0;p[1]<-2.11;
k<-v<-c()
#k[1]<-(w*p[1])/(z^2*p[1]+sige)
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+sigmae)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+sigmau
  v[t]<-y[t]-z*a[t-1]
  a[t]<-w*a[t-1]+k[t]*v[t]
}
matplot(cbind(y,alpha,a),type="l",main=expression(paste(y," ,  ", alpha," , ", a)))
```



Wonderful and very easy to implement using R!

Exercise: Consider the following model:

$$y_t = 1.05 * \alpha_{t-1} + e_t$$
$$\alpha_t = .5 + .8\alpha_{t-1} + u_t$$

Where $\sigma_e^2 = 2$ and $\sigma_u^2 = .3$. Now, using set.seed(321) do the following: (1) generate it (2) implement the Kalman filter (3) plot your results

Below we provide the function that generate a generic state-space model as in eq. 1 and another one that run the Kalman filter recursions as in 2:

```
StateSpaceGen<-function(param){
  sigmae<-param[[1]];sigmau=param[[2]];
  z<-param[[3]];w<-param[[4]];
  const<-param[[5]];
  n<-100
  e<-sqrt(sigmae)*rnorm(n)
  u<-sqrt(sigmau)*rnorm(n)
  y<-alpha<-c()
  y[1]=e[1];alpha[1]<-u[1]
  for(t in 2:n){
    y[t]<-z*alpha[t-1]+e[t]
    alpha[t]<-const+w*alpha[t-1]+u[t]
  }
```

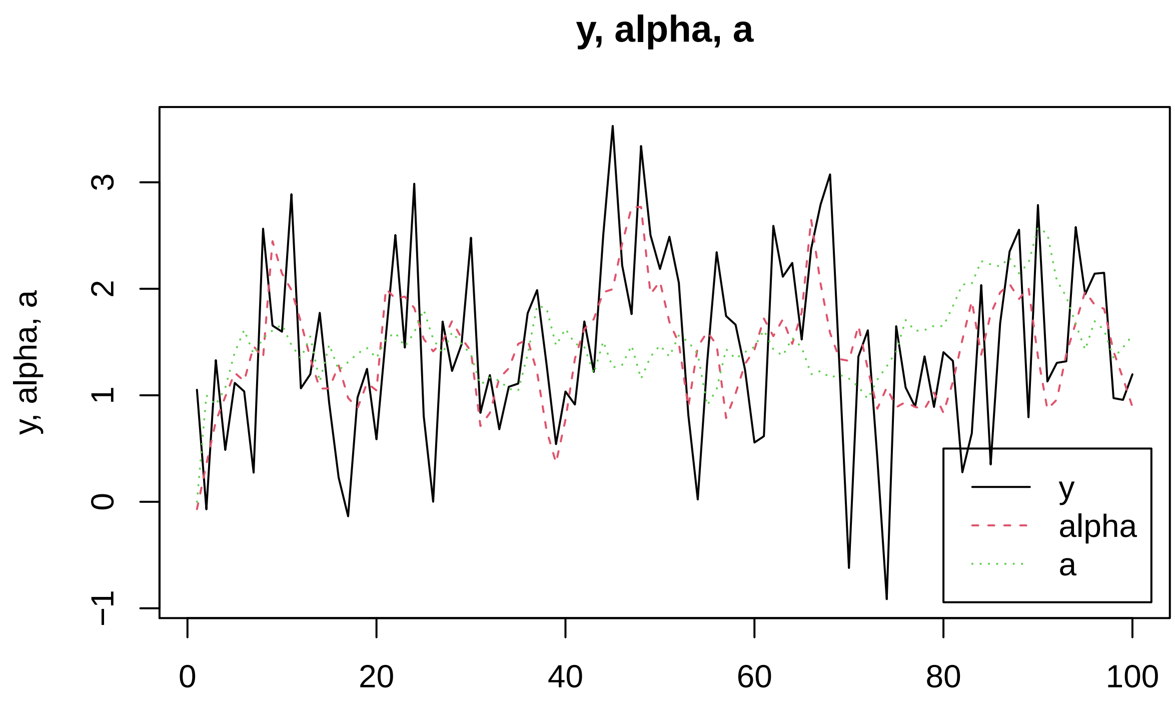
```r
  cbind(y,alpha)
}

KF<-function(param){
  sigmae<-param[[1]];sigmau<-param[[2]];
  z<-param[[3]];w<-param[[4]];
  const<-param[[5]];y<-param[[6]];
  a<-c();p<-c()
  a[1]=y[1];p[1]<-10000
  if(w<1){a[1]=0;p[1]<-sigmau/(1-w^2)}
  k<-v<-c()
  for(t in 2:n){
    k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+sigmae)
    p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+sigmau
    v[t]<-y[t]-z*a[t-1]
    a[t]<-const+w*a[t-1]+k[t]*v[t]
  };
  cbind(a,v,k,p)
}
```

Let's see an example here:

```r
set.seed(222)
matplot(cbind(StateSpaceGen(list(.5,.1,1,.8,.3)),KF(list(.5,.1,1,.8,.3,StateSpaceGen(list(.5,.1,1,.8,.3

mymat<-cbind(StateSpaceGen(list(.5,.1,1,.8,.3)),KF(list(.5,.1,1,.8,.3,StateSpaceGen(list(.5,.1,1,.8,.3)
colnames(mymat)<-c("y","alpha","a")
legend(x=80,y=0.5,legend = colnames(mymat), col = 1:3, lty = 1:3)
```



```
#Many thanks to Loop for pointing me this! :)
```

Likelihood function and model estimation

We have shown the derivation of the Kalman filter and its implementation assuming we know the variances of the errors. However, the question that now comes: what are the parameters to be chosen in order to make an estimate of $E(\alpha_t|Y_t)$ as closed as possible to $E(\alpha_t)$?

This is very important since in practice we do not know these variances and we need to make a guess (an estimation).

First of all, since y_t is a combination of normal variables, it has also a normal distribution. We can therefore write the Probability density function of y as follows:

$$Probability(y) = \frac{1}{\sqrt{(2\pi\sigma_y^2)}} exp(-\frac{1}{2}\frac{(y - E(\alpha|Y))^2}{\sigma_y^2}]$$

Secondly, since our aim is to forecast y_t, we are looking for the set of parameters that help us contructing a_t as closed as possible to y_t.

Now, as shown above, we have that the variance of y is $\sigma_y^2 = z^2 p_{t-1} + \sigma_e^2$. It is important to focus on the following difference $y_t - a_{t-1}$ that we defined as v_t, representing the “prediction error" when predicting y_t using a_{t-1}. Therefore we can write the probability distribution for a generic time y_t as:

$$Probability(y_t) = \frac{1}{\sqrt{2\pi(z^2 p_{t-1} + \sigma_e^2)}} exp(-\frac{1}{2}\frac{v_t^2}{(z^2 p_{t-1} + \sigma_e^2)})$$

Assuming that the observations y_t are independent, the joint distribution defined as likelihood function is:

$$Likelihood = Prob(y_1) \times Prob(y_2) \times Prob(y_3) \cdots Prob(y_n) = \prod_{t=1}^{n} Prob(y_t) = \left(\Pi_{t=1}^{n} 2\pi^{-\frac{1}{2}}(z^2 p_{t-1} + \sigma_e^2)^{-\frac{1}{2}}\right) exp(-\frac{1}{2}\sum_{t=1}^{n} \frac{v_t^2}{(z^2 p_{t-1}+\sigma_e^2)})$$

taking the logarithm of the likelihood we obtain:

$$\log L = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\sum_{t=1}^{n}\log(z^2 p_{t-1} + \sigma_e^2) - \frac{1}{2}\sum_{t=1}^{n}\frac{v_t^2}{(z^2 p_{t-1} + \sigma_e^2)} \quad (4)$$

Therefore, the set of parameters to be chosen in order to make an estimate of $E(\alpha_t|Y_t)$ as closed as possible to y_t is the one that maximize the log L function. Or alternatively, the set that minimize minus the log L function.

This chunck below provides an example:

```r
n<-100
set.seed(1)
su<-.05
se<-.5
e<-sqrt(se)*rnorm(n)
u<-sqrt(su)*rnorm(n)
z<-1;wreal<-.86
const<-.6
y<-alpha<-c()
y[1]=const+e[1];alpha[1]<-const+u[1]
for(t in 2:n){
  y[t]<-z*alpha[t-1]+e[t]
  alpha[t]<-const+wreal*alpha[t-1]+u[t]
}
########### standard Kalman filter approach##################
a<-c();p<-c()
a[1]=0;p[1]<-10;
k<-v<-c()
fu<-function(mypa){w<-abs(mypa[1]);se<-abs(mypa[2]);
su<-abs(mypa[3]);co<-abs(mypa[4]);
  z<-1
  likelihood<-0
  for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+se)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+su
  v[t]<-y[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
 likelihood<-likelihood+.5*log(2*pi)+.5*log(z^2*p[t-1]+se)+.5*(v[t]^2/(z^2*p[t-1]+se))
}
likelihood
}
results<-optim(c(.85,.5,.3,.3),fu)
print("The results of the standard KF approach")
```

```
## [1] "The results of the standard KF approach"
```

```r
results[[1]]
```

```
## [1] 0.81408439 0.37997726 0.05017142 0.79121855
```

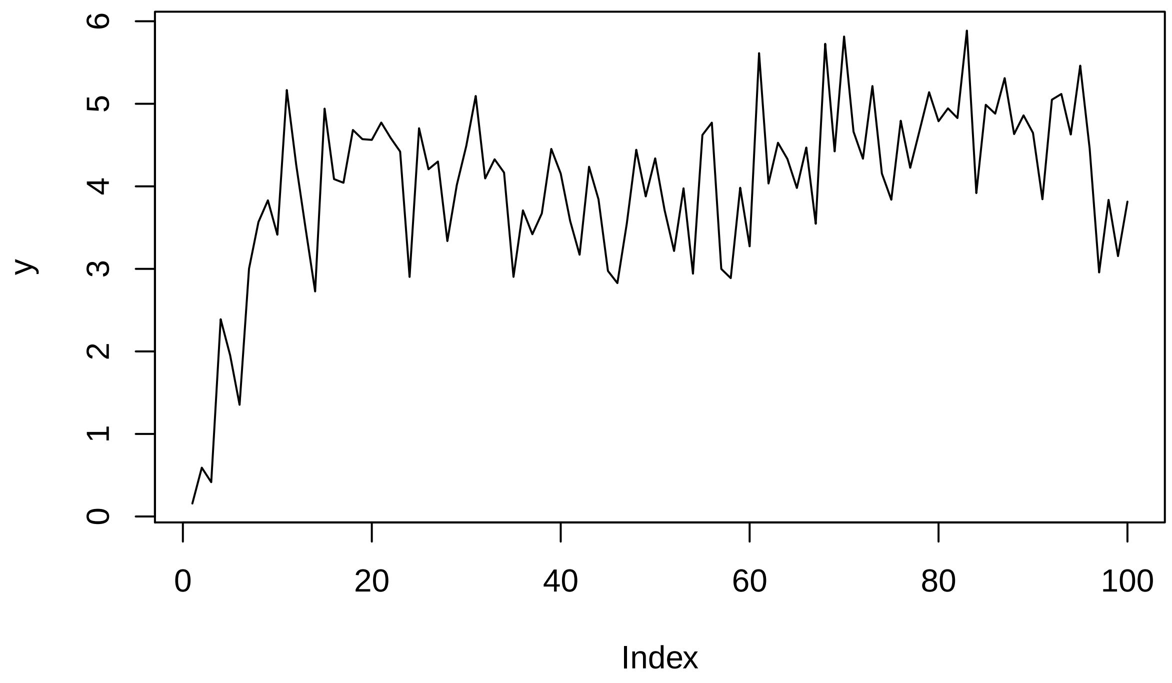
```r
print("The true parameters")
```

```
## [1] "The true parameters"
```

```r
cbind(wreal,se,su,const)
```

```
##      wreal  se   su const
## [1,]  0.86 0.5 0.05   0.6
```

```r
plot(y,type = "l")
```



One potential issue of using the Log-likeliood as above is related with the number of parameters. Indeed, the likelihood is a single value and gets maximized by all parameters. Therefore the less parameters we have, the more efficient the estimation. In the univariate framework, one trick that we can use is to concentrate out the likelihood function.

Rewrite the log-Likelihood above as follows:

$$\log L = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\sum_{t=1}^{n}\log[(z^2\tilde{p}_{t-1}+1)\times\sigma_e^2] - \frac{1}{2}\sum_{t=1}^{n}\left(\frac{v_t^2}{[(z^2\tilde{p}_{t-1}+1)\times\sigma_e^2]}\right)$$

Where $\tilde{p}_{t-1}\sigma_e^2 = p_{t-1}$. Now defining $\frac{\sigma_u^2}{\sigma_e^2} = q$, the Kalman filter recursions for this reparametrized model can be written as:

$$\begin{aligned}\tilde{p}_t &= w^2\tilde{p}_{t-1} - wzk_t\tilde{p}_{t-1} + q\\ k_t &= \frac{zw\tilde{p}_{t-1}}{z^2\tilde{p}_{t-1}+1}\\ a_t &= c + wa_{t-1} + k_t\left(y_t - za_{t-1}\right)\\ v_t &= (y_t - za_{t-1})\end{aligned} \tag{5}$$

Interestingly, k_t is not affected by the fact that we consider $\tilde{p}_t$ in place of p_t, neither are a_t and v_t.

Now note that $\tilde{\sigma}_e^2 = \frac{1}{(n-1)}\sum_{t=2}^{n}\left(\frac{v_t^2}{z^2\tilde{p}_{t-1}+1}\right)$ represents a consistent estimator of σ_e^2. Therefore the concentrated log-Likelihood gets:

$$\log L = -\frac{n}{2}[\log(2\pi)+1] - \frac{1}{2}\sum_{t=1}^{n}\log(z^2\tilde{p}_{t-1}+1) - \frac{n}{2}\log[\tilde{\sigma}_e^2]$$

Thererfore, one can now choose the set of parameters (q; w; z ; c) that maximize the last expression. Once the set is estimated, we can estimate all the variances as follows: $\tilde{\sigma}_e^2 = \frac{1}{(n-1)}\sum_{t=2}^{n}\left(\frac{v_t^2}{z^2\tilde{p}_{t-1}+1}\right)$ and $\tilde{\sigma}_u^2 = q*\tilde{\sigma}_e^2$

Initializing the Kalman filter recursions

These recursions need to be initialized. In particular, when we are at time $t = 1$ we need to determine the value of a_1. In principle the choice of this value depends on the type of process followed by x_t. For example, when x_t follows an AR(1) process (i.e. $0 \leq w \leq 1$) , the best choice is $a_1 = 0$ since the process is mean reverting. On the other hand, when x_t is a random walk (i.e. $w = 1$) it is better to use $a_1 = y_1$. We do not consider the case when $w > 1$ since this would be an explosive autoregressive process. Finally, when using the standard Kalman filter recursions, one should also initialize p_t. For the AR(1) case, as shown above, one can initialize as follows $p_1 = \frac{\sigma_u^2}{1-\omega^2}$. When $\omega = 1$, as in the random walk process, one should use $p_1 = \infty$. Indeed, the so called diffuse initialization proposes, for the case considered, $p_1 = 10000$, an arbitrary high number.

Concentrated Log-likelihood in action!

The code below generate a model and estimate it using the Concentrated Log-likelihood with (2):

```r
n<-100
set.seed(61)
su<-.1
se<-.4
qreal<-su/se
e<-sqrt(se)*rnorm(n)
u<-sqrt(su)*rnorm(n)
z<-1;wreal<-.97
```

```r
y<-alpha<-c()
y[1]=e[1];alpha[1]<-u[1]
for(t in 2:n){
  y[t]<-z*alpha[t-1]+e[t]
  alpha[t]<-wreal*alpha[t-1]+u[t]
}
########### standard Kalman filter approach##################
a<-c();p<-c()
a[1]=0;p[1]<-10;
k<-v<-c()
fu<-function(mypa){w<-abs(mypa[1]);q<-abs(mypa[2]);
  z<-1
  likelihood<-0
  sigmae<-0
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-y[t]-z*a[t-1]
  a[t]<-w*a[t-1]+k[t]*v[t]
   sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
  likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
}
likelihood+.5*n*log(sigmae/n)
}
results<-optim(c(.85,.5),fu)
print("The results of the standard KF approach")
```

```
## [1] "The results of the standard KF approach"
```

```r
results[[1]]
```

```
## [1] 0.9883714 0.1616306
```

```r
print("The true parameters")
```

```
## [1] "The true parameters"
```

```r
cbind(wreal,qreal)
```

```
##      wreal qreal
## [1,]  0.97  0.25
```

One can see that the estimates are rather closed to the true parameters.

State-Space models and the Kalman filter in action!

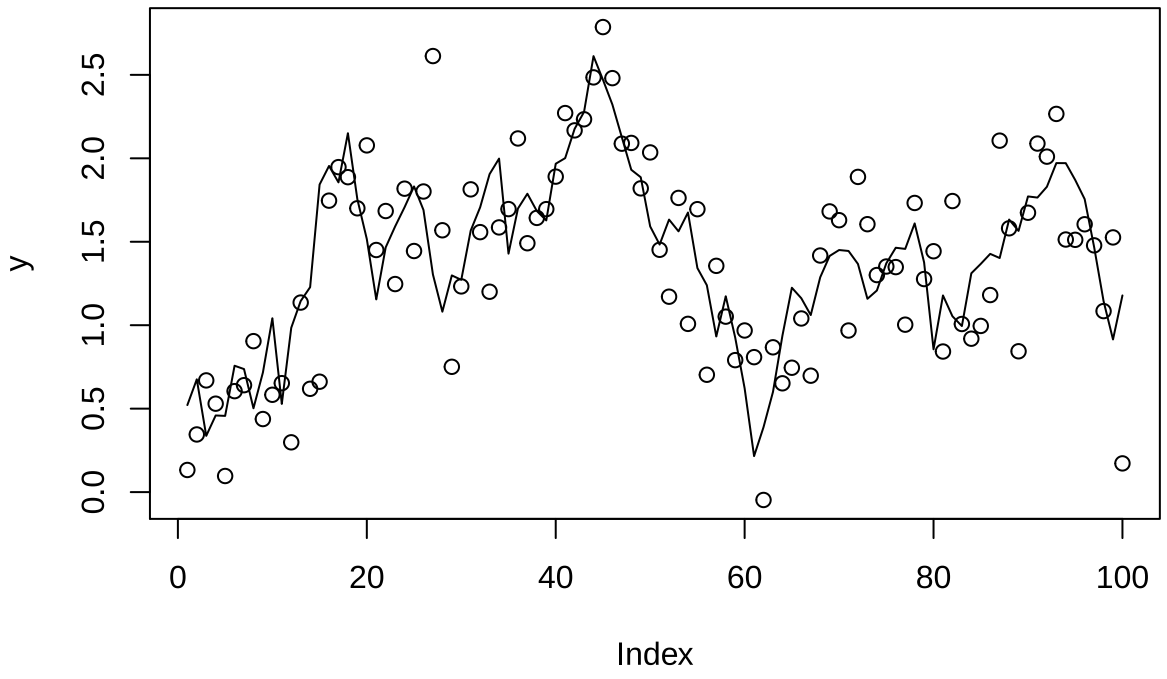
We can now stop with the theory and start getting fun using *R* and reproducing the results we obtained above.

For example suppose we want to generete the following state-space model:

$$y_t = \alpha_{t-1} + e_t$$
$$\alpha_t = .2 + .85\alpha_{t-1} + u_t$$

where $e \sim Normal(\mu_e = 0; \sigma_e^2 = .1)$ and $u \sim Normal(\mu_u = 0; \sigma_u^2 = .05)$ where $n = 100$. In addition suppose we want to estimate the parameters of the model.

```
n<-100
set.seed(1265)
e<-sqrt(.1)*rnorm(n)
u<-sqrt(.05)*rnorm(n)
constant<-.2
y<-alpha<-c()
y[1]=e[1];alpha[1]<-u[1]
for(t in 2:n){
  y[t]<-alpha[t-1]+e[t]
  alpha[t]<-constant+.85*alpha[t-1]+u[t]
  }
plot(y,type="p")
  lines(alpha,type="l")
```



Using the standard KF we obtain the following results:

```
########### standard Kalman filter approach##################
a<-c();p<-c()
a[1]=0;p[1]<-1;
```

```r
k<-v<-c()
z<-1
fu<-function(mypa){w<-abs(mypa[1]);#z<-abs(mypa[2]);
q<-abs(mypa[2]);co<-abs(mypa[3])
    likelihood<-sigmae<-0
    for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-y[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
  likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
}
  likelihood+.5*n*log(sigmae/n)
}
results<-optim(c(.9,1,.1),fu)
v[1]<-0
w<-abs(results[[1]][[1]]);#z<-abs(results[[1]][[2]]);
q<-abs(results[[1]][[2]]);co<-abs(results[[1]][[3]])
likelihood<-sigmae<-0
for(t in 2:length(y)){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-y[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+v[t]^2/(z^2*p[t-1]+1)
  likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
}
likelihood<-likelihood+.5*n*log(sigmae/n)
sigmae<-sigmae/length(y)
sigmau<-q*sigmae
cbind(co,w,z,sigmae,sigmau)
```

```
##              co         w z     sigmae     sigmau
## [1,] 0.1748028 0.8775725 1 0.09778626 0.04862576
```

One can see that the estimates are rather precise.

Exercise

Create a function that run the KF and the maximize the likelihood returning the estimated parameters.

The Local level model (or simple exponential smoothing)

The local level is one of the simplest state-space models. This model assumes $w = z = 1$ and $constant = 0$, such that we have:

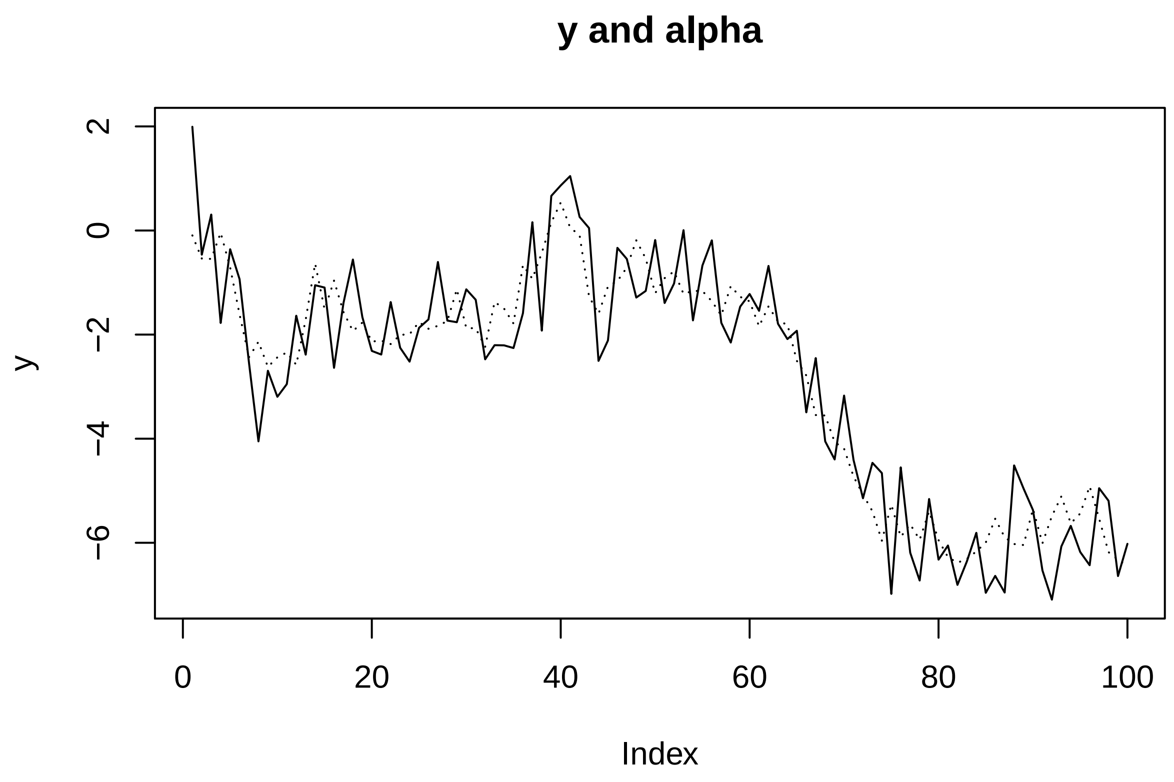
$$y_t = \alpha_{t-1} + e_t$$
$$\alpha_t = \alpha_{t-1} + u_t$$

As a consequence one has only one parameter to estimate q. This is the only ingredient that maximize the concentreted log Likelihood function.

For example suppose we want to generete the following state-space model:

where $e \sim Normal(\mu = 0; \sigma_e^2 = .5)$ and $u \sim Normal(\mu = 0; \sigma_u^2 = .2)$ where $n = 100$. Here q is 0.4.

```
set.seed(153)
n<-100
e<-sqrt(.5)*rnorm(n)
u<-sqrt(.2)*rnorm(n)
y<-alpha<-c()
y[1]=e[1];alpha[1]<-u[1]
for(t in 2:n){
  y[t]<-alpha[t-1]+e[t]
  alpha[t]<-alpha[t-1]+u[t]
}
plot(y,type="l",main="y and alpha")
  lines(alpha,type="l",lty=3)
```



We can now estimate this model with the KF recursions. The code below estimate the model with the 4 recursions.

```
########### standard Kalman filter approach###################
a<-c();p<-c()
a[1]=y[1];p[1]<-10000;
k<-v<-c()
fu<-function(mypa){q<-abs(mypa);
  z<-w<-1
  likelihood<-sigmae<-0
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-y[t]-z*a[t-1]
  a[t]<-w*a[t-1]+k[t]*v[t]
    sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
  likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
}
  likelihood+.5*n*log(sigmae/n)
}
results<-optim(c(.2),fu,method = "Brent", lower = 0, upper = 1)
results[[1]]
```

```
## [1] 0.5110821
```

We now derive the estimates of the parameters (the two variances)

```
z<-w<-1
q<-results[[1]]
  sigmae<-0
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-y[t]-z*a[t-1]
  a[t]<-w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+v[t]^2/(z^2*p[t-1]+1)
}
#This is the variance of e
    sigmae/(n-1)
```

```
## [1] 0.4632777
```

```
#This is the variance of u
  q*(sigmae/(n-1))
```

```
## [1] 0.2367729
```

Exercise: (Auto)regression with time varying parameter

Consider the following model:

$$\begin{aligned} y_t &= y_{t-1}\beta_{t-1} + e_t \\ \beta_t &= \beta_{t-1} + u_t \end{aligned} \tag{6}$$

This model is a variant of model (1). Now try generate it and estimate it using the standard Kalman filter approach.

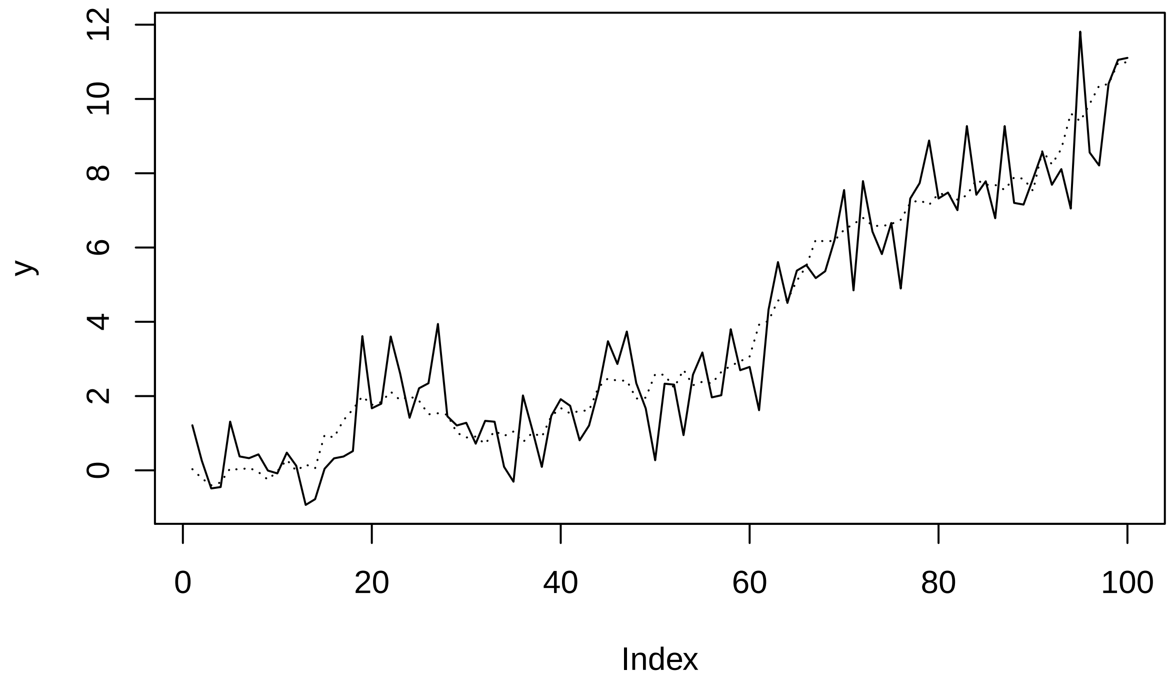
The Local Level with drift: (the Theta method)

This model is a simple variant of the local level but it is quite popular among the forecasters since it competes very well with other strong benchmark. This is also a reparametrization of the so-called Theta method as shown in (TOADD).

For example suppose we want $e \sim Normal(\mu = 0; \sigma^2_e = .8)$, $u \sim Normal(\mu = 0; \sigma^2_u = .1)$ and $constant = .1$, where $n = 100$. Here q is 0.125. This code generates the local level with drift:

```r
set.seed(572)
n<-100
e<-sqrt(.8)*rnorm(n)
u<-sqrt(.1)*rnorm(n)
y<-alpha<-c()
co<-.1
y[1]=e[1];alpha[1]<-u[1]
for(t in 2:n){
  alpha[t]<-co+alpha[t-1]+u[t]
  y[t]<-alpha[t-1]+e[t]

}
plot(y,type="l")
  lines(alpha,type="l",lty=3)
```



The solid line is the observed data y_t while the dotted line is the state α_t. We now estimate the model maximizing the concentrated log-likelihood using the KF:

```r
########### Kalman filter##################
a<-c();p<-c()
a[1]=y[1];p[1]<-10000;
k<-v<-c()
v[1]<-0
funcTheta<-function(parameters){q<-abs(parameters[1]);co<-abs(parameters[2]);
  z<-w<-1
  likelihood<-sigmae<-0
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-y[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
  likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
}
  likelihood+.5*n*log(sigmae/n)
}
results<-optim(c(.6,.2),funcTheta)
q<-abs(results[[1]][[1]]);co<-abs(results[[1]][[2]]);
  z<-w<-1
      sigmae<-0
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-y[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+v[t]^2/(z^2*p[t-1]+1)
  }


#This is the variance of e
    sigmae/(n-1)
```

```
## [1] 0.8100567
```

```r
#This is the variance of u
  q*(sigmae/(n-1))
```

```
## [1] 0.152386
```

Suppose we want to create a function that generate a time series model and a function that estimate the model using the steady-stae approach, say the Theta model, and return the estimates of the parameters, the estimates of a_t and the prediction error v_t.

The first function can be created like that:

```r
generateTheta<-function(n,sigmae,sigmau,co){
e<-sqrt(sigmae)*rnorm(n)
u<-sqrt(sigmau)*rnorm(n)
y<-alpha<-c()
y[1]=e[1];alpha[1]<-u[1]
for(t in 2:n){
  alpha[t]<-co+alpha[t-1]+u[t]
  y[t]<-alpha[t-1]+e[t]

}
y
}
```

The second function can be created like that:

```
EstimateTheta<-function(y){
  n<-length(y)
a<-c();p<-c()
a[1]=0;p[1]<-0;
k<-v<-c()
fu<-function(mypa){q<-abs(mypa[1]);co<-(mypa[2])
    likelihood<-sigmae<-0
w<-z<-1
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-y[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
  likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
}
  likelihood+.5*n*log(sigmae/n)
}
results<-optim(c(.5,.2),fu)
v[1]<-0
w<-z<-1
q<-abs(results[[1]][[1]]);co<-(results[[1]][[2]])
sigmae<-0
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-y[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+v[t]^2/(z^2*p[t-1]+1)
}
sigmae<-sigmae/length(y)
sigmau<-q*sigmae
thelist<-list(cbind(sigmae,sigmau,co),a,v);
thelist
}
```

Let's test if they work!

```
set.seed(11)
EstimateTheta(generateTheta(100,.6,.2,1))[[1]]
```

```
##          sigmae    sigmau       co
## [1,] 0.5786558 0.1616449 1.050598
```

Exercise Generate & Estimate

Create a function, define it GenerateSS, that generate the model (1) imposing z=1.

Create a function that takes the data and estimate model (1) imposing z=1. The function has to return a list containing (1) the estimates of the parameters (2) the prediction errors (3) the estimates of $a_t = E(\alpha_t|Y)$

Single Source of Error approach

The literature in the last two decades has focused the attention on the state-space model with just one source of error.

Consider the following State-Space model:

$$\begin{aligned} y_t &= z\alpha_{t-1} + e_t \\ \alpha_t &= c + w\alpha_{t-1} + \gamma e_t \end{aligned} \tag{7}$$

This model differs from (1) since there is only one noise determining both the state-equation and the observations. One important feature with this approach is that we do not need to run all the four KF recursions but only two, that is :

$$\begin{aligned} e_t &= y_t - za_{t-1} \\ a_t &= c + wa_{t-1} + \gamma e_t \end{aligned} \tag{8}$$

The likelihood function for model (7) can be derived easily. First of all, when e_t is Normal then also y_t is. We can therefore write the Probability density function of y as follows:

$$Probability(y) = \frac{1}{\sqrt{(2\pi\sigma_y^2)}} exp(-\frac{1}{2}\frac{(y - za_{t-1})^2}{\sigma_y^2}) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp(-\frac{1}{2}\frac{e_t^2}{\sigma_y^2})$$

The main difference with the framework discussed above is that the prediction error estimated using $e_t = y_t - za_{t-1}$ corresponds to the error in the observational equation.

Assuming that the observations y_t are independent, the joint distribution defined as likelihood function is:

$$Likelihood = Prob(y_1) \times Prob(y_2) \times Prob(y_3) \cdots Prob(y_n) = \prod_{t=1}^{n} Prob(y_t) = \\ \Pi_{t=1}^{n}(2\pi^{-\frac{1}{2}}\sigma_y^2)^{-.5} exp\left(\sum_{t=1}^{n} -\frac{1}{2}\frac{e_t^2}{\sigma_y^2}\right)$$

The log-Likelihood function for this representation is the following

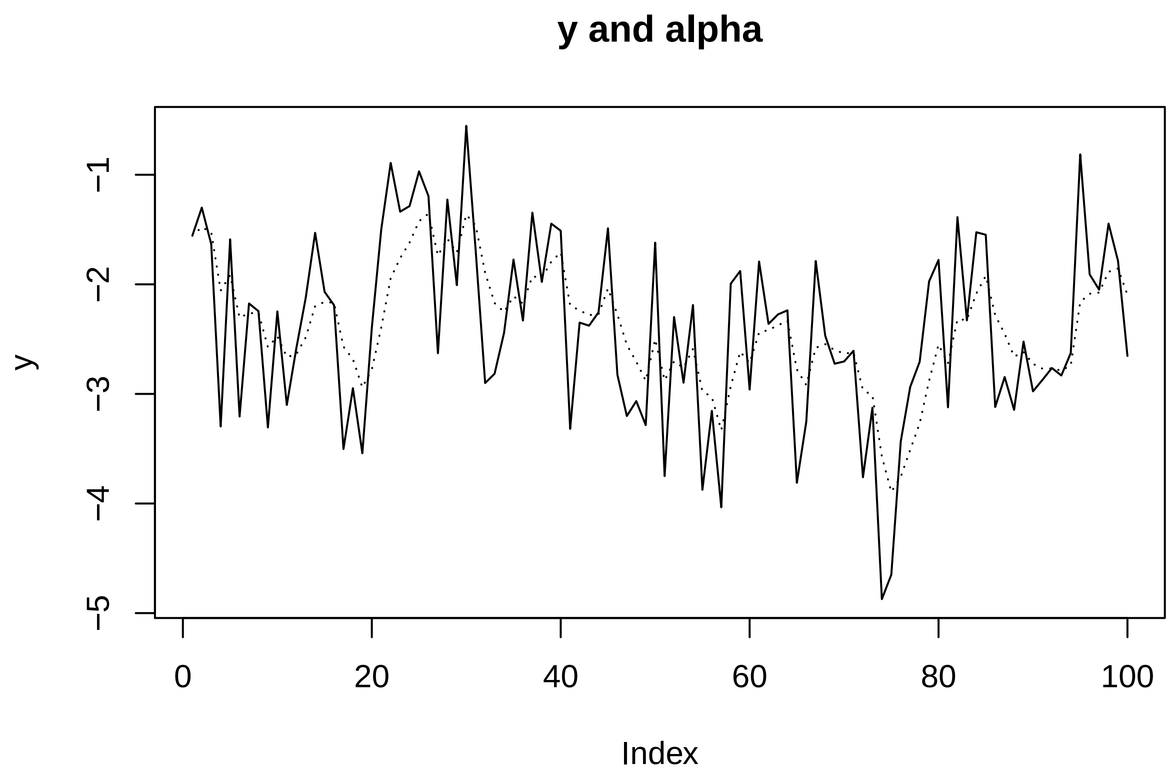
$$\log L = -\frac{n}{2}\log(2\pi\sigma_y^2) - \frac{1}{2}\sum_{t=1}^{n}\frac{e_t^2}{\sigma_y^2} \tag{9}$$

Note that the likelihood can be maximized by just minimizing $\sum e_t^2$. This is very simple!

The exponential smoothing with one source of error

Lets see the single source of error is practice. Below I generate an exponential smoothing model:

```r
set.seed(213)
n<-100
e<-sqrt(.6)*rnorm(n)
gamma<-.3
y<-alpha<-c()
y[1]=e[1];alpha[1]<-e[1]
for(t in 2:n){
  y[t]<-alpha[t-1]+e[t]
  alpha[t]<-alpha[t-1]+gamma*e[t]
}
plot(y,type="l",main="y and alpha")
  lines(alpha,type="l",lty=3)
```



We can now estimate this model with the two recursions.

```r
a<-c()
a[1]=y[1];
e<-matrix(0,length(y),1)
fu<-function(mypa){gamma<-abs(mypa);
  for(t in 2:n){
  e[t]<-y[t]-z*a[t-1]
  a[t]<-a[t-1]+gamma*e[t]
```

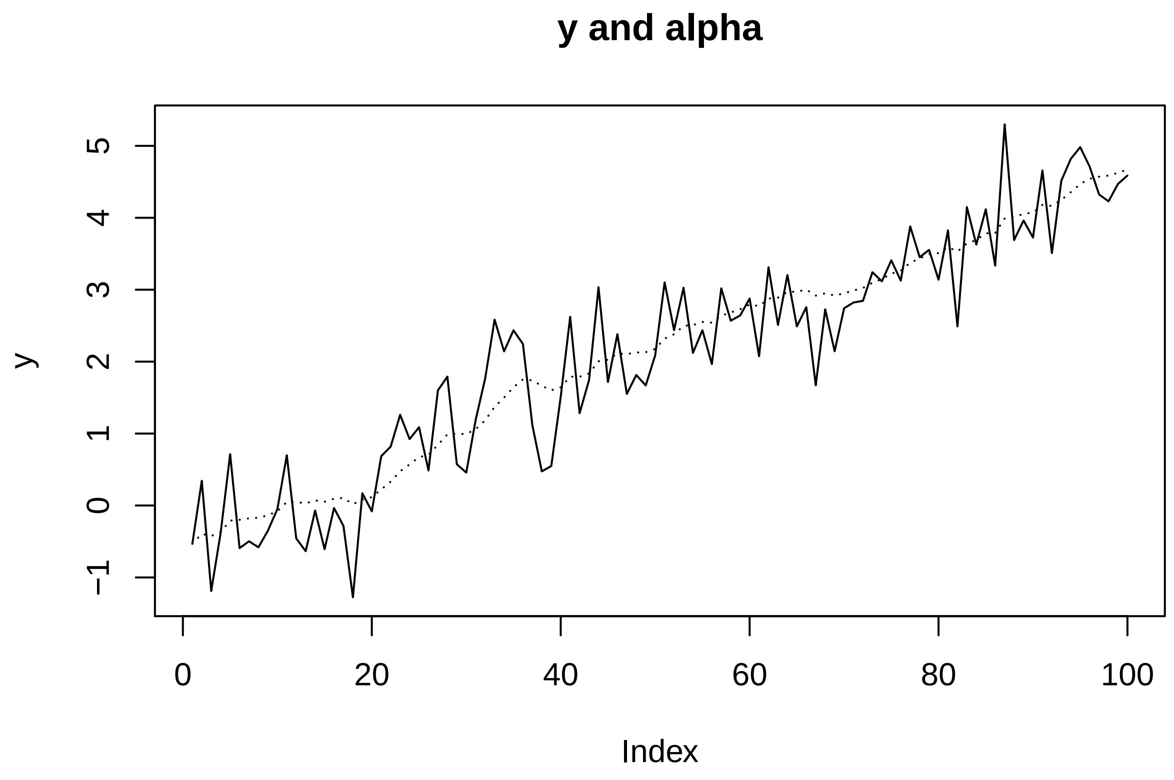
```
    }
  sum(e^2)/n
}
results<-optim(c(.2),fu,method = "Brent", lower = 0, upper = 1)
results[[1]]
```

```
## [1] 0.3333491
```

The Theta method with one source of error

Lets see the single source of error is practice. Below I generate an exponential smoothing model:

```
set.seed(5)
n<-100
e<-sqrt(.4)*rnorm(n)
gamma<-.1
con<-.05
y<-alpha<-c()
y[1]=e[1];alpha[1]<-e[1]
for(t in 2:n){
  y[t]<-alpha[t-1]+e[t]
  alpha[t]<-con+alpha[t-1]+gamma*e[t]
}
plot(y,type="l",main="y and alpha")
  lines(alpha,type="l",lty=3)
```



We can now estimate this model with the two recursions.

```r
a<-c()
a[1]=y[1];
e<-matrix(0,length(y),1)
fu<-function(mypa){gamma<-abs(mypa[1]);
co<-abs(mypa[2]);
  for(t in 2:n){
  e[t]<-y[t]-z*a[t-1]
  a[t]<-co+a[t-1]+gamma*e[t]
    }
  sum(e^2)/n
}
results<-optim(c(.2,.1),fu)
results[[2]]
```

```
## [1] 0.345617
```

```r
results[[1]]
```

```
## [1] -6.366368e-09  5.325348e-02
```

Note that the estimation returns the variance of the noise as well as the γ and the constant.

Exercise: Generate the following model and estimate it using set.seed(1452): $y_t = a_{t-1} + e_t$ **with** $a_t = 0.94 \times a_{t-1} + .2e_t$ **where** e_t **is normal with variance** $\sigma_e^2 = 1.2$

Exercise: the damped trend model in the SSOE –> Consider the following model:

$$\begin{aligned} y_t &= \alpha_{t-1} + \phi\beta_{t-1} + e_t \\ \alpha_t &= \alpha_{t-1} + \phi\beta_{t-1} + \gamma e_t \\ \beta_t &= \phi\beta_{t-1} + \theta e_t \end{aligned} \tag{10}$$

Now generate it such that $\sigma_e^2 = .5$**;** $\gamma = .6$**;** $\phi = .93$**;** $\theta = .2$ **using set.seed(123). Finally estimate it and provide the results**

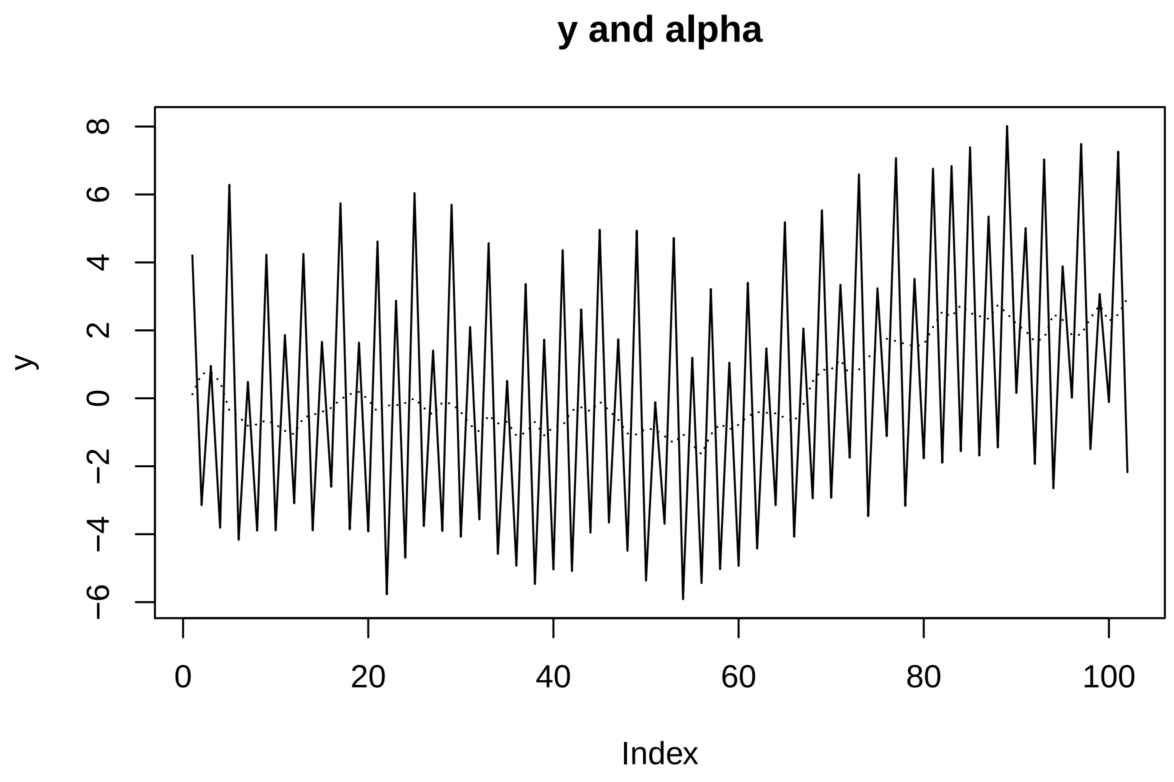
Seasonality

Sometimes data shown a dynamic behavior that tends to repeat from time to time depending on the specific time frequency. In this case, the time series are affected by a seasonal component representing a non-negligiable feature of the time series dynamics. In this section we introduce two simple procedures that can be used in order to "clean" the series from this seasonal component. We first introduce the additive case and then the multiplicative case below.

Additive seasonality

Suppose we have a quarterly series, that is a series observed 4 times per year, like this:

```r
set.seed(1213)
n<-102
e<-sqrt(.5)*rnorm(n)
u<-sqrt(.1)*rnorm(n)
y<-alpha<-c()
seasfactor<-c(5,-4,2,-3)
s<-4
  seasonal<-rep(seasfactor,ceiling(n/s))[1:n]
y[1]=e[1]+seasonal[1];alpha[1]<-u[1]
for(t in 2:n){
  y[t]<-seasonal[t]+alpha[t-1]+e[t]
  alpha[t]<-alpha[t-1]+u[t]
}
plot(y,type="l",main="y and alpha")
  lines(alpha,type="l",lty=3)
```



One way to threat this series is to remove the seasonal component by using the so called moving average appraoch. These are the steps:

Take the centered moving average of the series, call it CMA_t

Subtract the CMA from the original series $residuals_t = y_t - CMA_t$

Average the elements of $residuals_t$ by season and obtain the seasonal factor

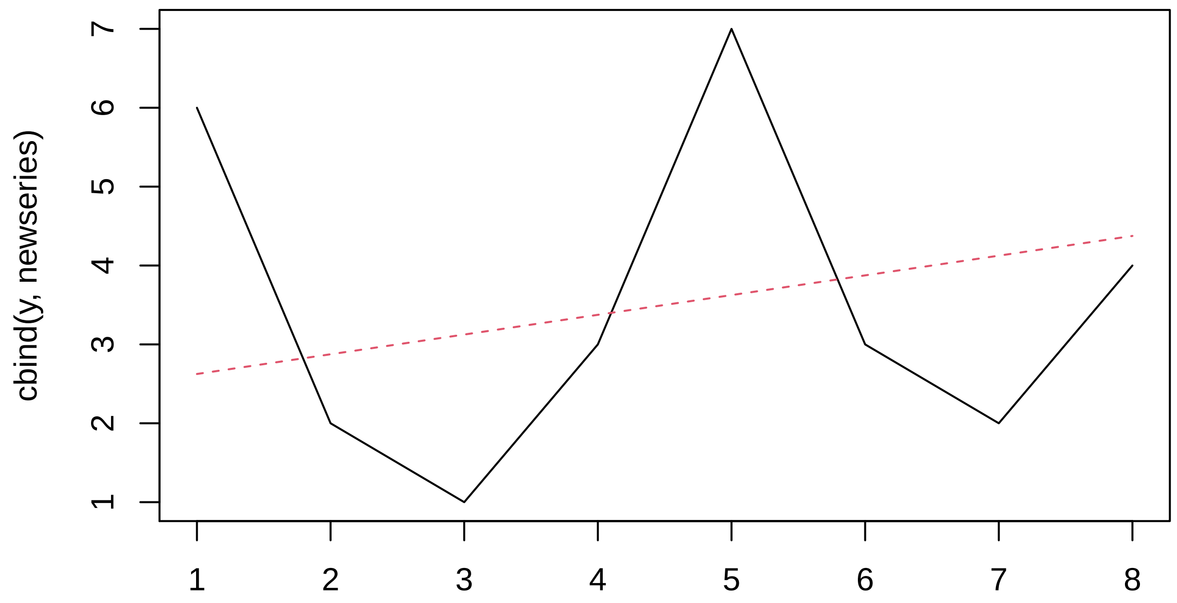
Subtract the element of y_t by the corresponding seasonal factor

Step one requires to create a series of average values centered at time t. Define s the number of seasons considered. In the quarterly case we have 4 values. If we want to create a centered moving average we need to select values at time t , $t \pm 1$, $\cdots \pm \frac{s}{2}$. In the quarterly case we need to take 5 values t , $t \pm 1$, $t \pm 2$, but we want a moving average of 4 elements! So how to create this average? In the quarterly case for example we can take t , $t \pm 1$, $\frac{t \pm 2}{2}$, that is half of the extreme values $\pm\frac{t \pm 2}{2}$. In other terms we want $\frac{0.5 * x_{t-2} + x_{t-1} + x_t + x_{t+1} + 0.5 * x_{t+2}}{4}$. The next code run an example of the procedure to deseasonalise a short series. The code is not smart but makes the trick:

```r
y<-c(6,2,1,3,7,3,2,4)
cma<-matrix(NA,length(y),1)
cma[3]<-(.5*y[1]+y[2]+y[3]+y[4]+.5*y[5])/4
cma[4]<-(.5*y[2]+y[3]+y[4]+y[5]+.5*y[6])/4
cma[5]<-(.5*y[3]+y[4]+y[5]+y[6]+.5*y[7])/4
cma[6]<-(.5*y[4]+y[5]+y[6]+y[7]+.5*y[8])/4
residuals=y-cma
cbind(y,cma,residuals)
```

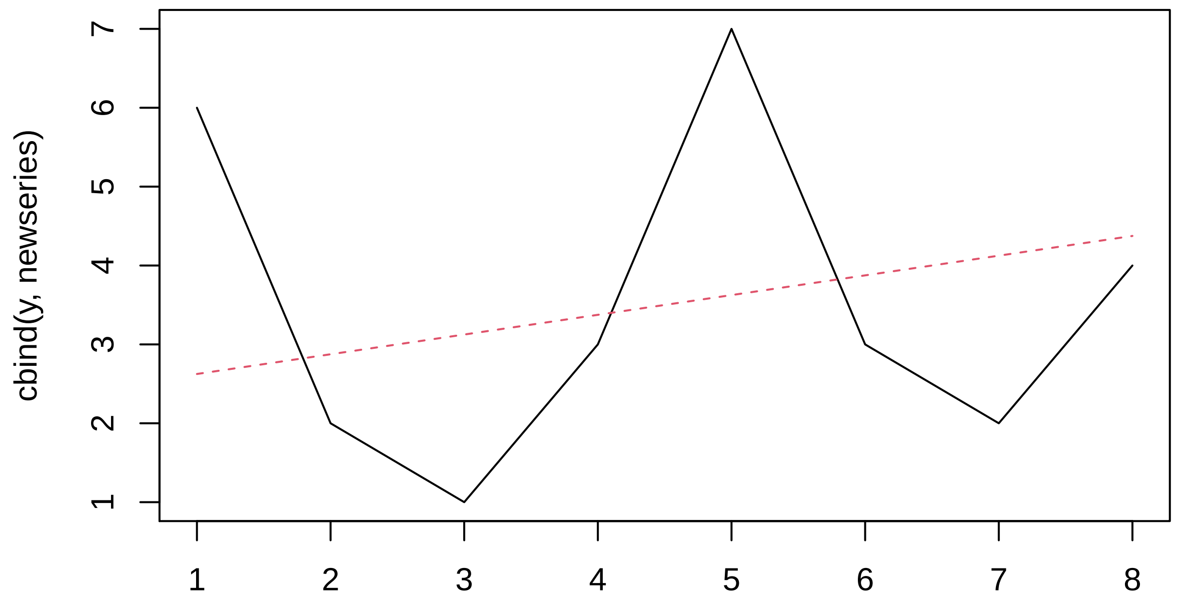
```
##      y
## [1,] 6    NA     NA
## [2,] 2    NA     NA
## [3,] 1 3.125 -2.125
## [4,] 3 3.375 -0.375
## [5,] 7 3.625  3.375
## [6,] 3 3.875 -0.875
## [7,] 2    NA     NA
## [8,] 4    NA     NA
```

```r
factors<-cbind(mean(na.omit(residuals[c(1,5)])),mean(na.omit(residuals[c(2,6)])),
               mean(na.omit(residuals[c(3,7)])),mean(na.omit(residuals[c(4,8)])))
newseries<-y-rep(factors,2)
matplot(cbind(y,newseries),type="l")
```



The code makes what we want but it is not elegant and need to be generalized! Lets create a smarter code as below:

```
s<-4 # s is my frequency (for example: quarterly=4;monthly=12;weekly=52)
n<-length(y)
#This create the weights to be used in the moving average
w<-rep(1/(2*s),s+1);w[2:s]<-1/s
#This create the centered moving average vector
cma<-matrix(NA,length(y),1);
#This calculate the centered moving averag
for(g in 1:(length(y)-s)){cma[g+s/2]<-sum(w*y[g:(g+s)])};
#This is the residuals
residuals<-y-cma
#this creates the s factors as we want
factors<-c();for(seas in 1:s){
  factors[seas]<-mean(na.omit(residuals[seq(seas,length(y)-s+seas,by=s)]))}
#This allows to demean the factors variable
factors<-factors-rep(mean(factors),s)
#this is the last step: we take out the seasonal component
  newseries<-y-rep(factors,ceiling(n/s))[1:n]
matplot(cbind(y,newseries),type="l")
```

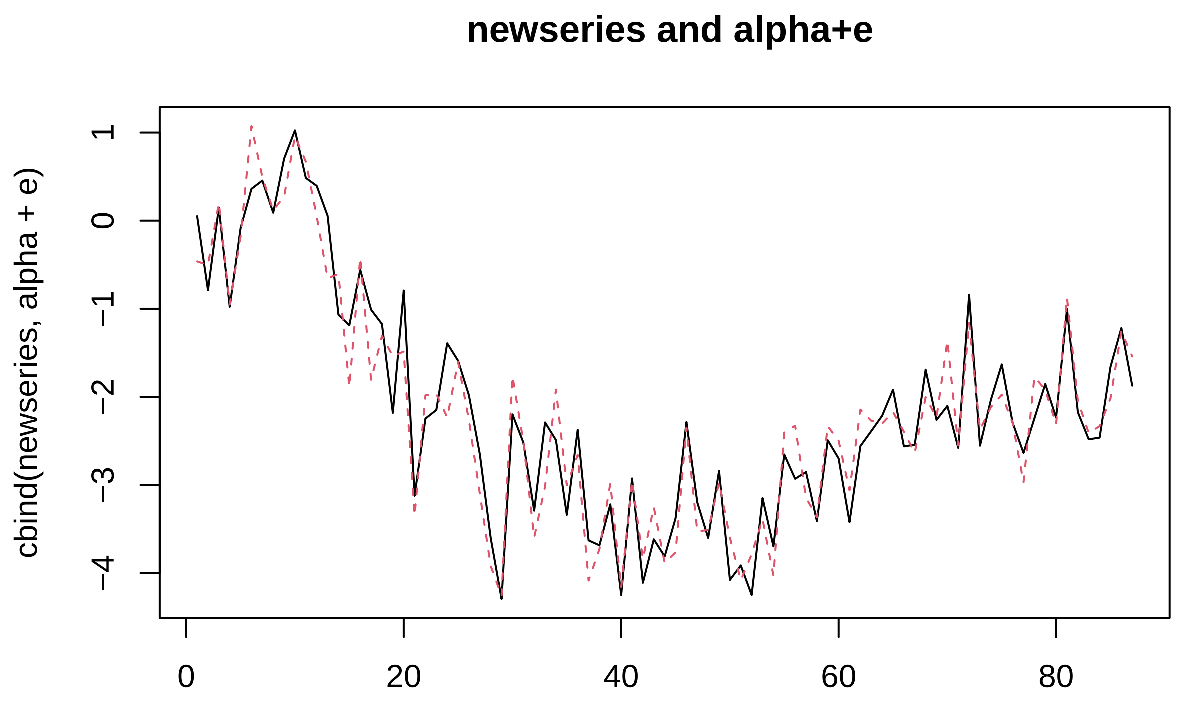


Lets generate a series and we then implement the process of taking out the seasonal component:

```r
set.seed(243)
n<-87
e<-sqrt(.3)*rnorm(n)
u<-sqrt(.1)*rnorm(n)
y<-alpha<-c()
seasfactor<-c(5,-4,2,-3)
s<-4
seasonal<-rep(seasfactor,ceiling(n/s))[1:n]

y[1]=e[1]+seasonal[1];alpha[1]<-u[1]
for(t in 2:n){
  y[t]<-seasonal[t]+alpha[t-1]+e[t]
  alpha[t]<-alpha[t-1]+u[t]
}

w<-rep(1/(2*s),s+1);w[2:s]<-1/s
cma<-matrix(NA,length(y),1);
for(g in 1:(length(y)-s)){cma[g+s/2]<-sum(w*y[g:(g+s)])};
residuals<-y-cma
factors<-c();for(seas in 1:s){
factors[seas]<-mean(na.omit(residuals[seq(seas,length(y)-s+seas,by=s)]))}
factors<-factors-rep(mean(factors),s)
newseries<-y-rep(factors,ceiling(n/s))[1:n]
matplot(cbind(newseries,alpha+e),type="l",main=" newseries and alpha+e")
```



It is interesting to note that the factors are very similar to the ones that generated the series. Indeed we have:

```r
print(factors)
```

```
## [1]  4.767247 -3.860166  2.109801 -3.016882
```

```r
print(seasfactor)
```

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

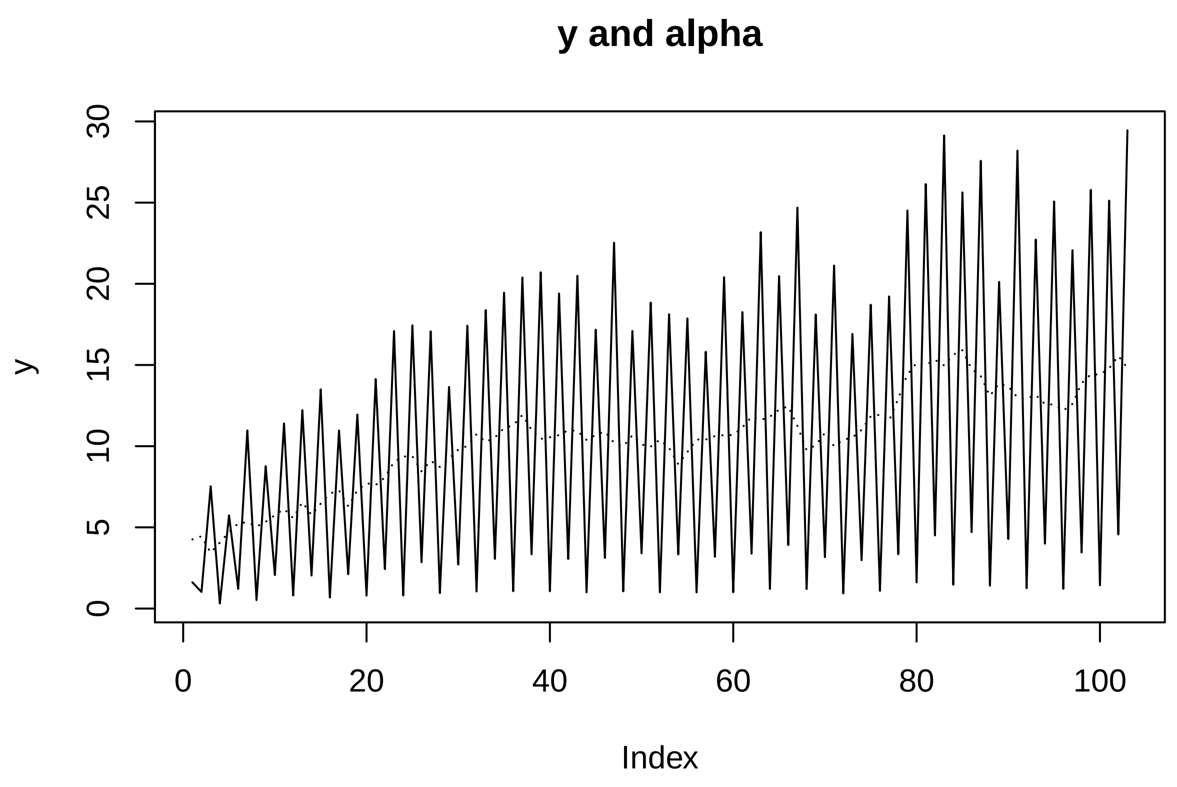
So this procedure allows to extract the seasonal component from a series. Note that a similar procedure can be run for the case where the seasonal component is multiplicative rather than additive. Below we show the procedure.

Multiplicative seasonality

Suppose we have a quarterly series, that is a series observed 4 times per year, like this:

```r
set.seed(7)
n<-103
e<-sqrt(.5)*rnorm(n)
u<-sqrt(.4)*rnorm(n)
```

```r
y<-alpha<-c()
seasfactor<-c(1.7,.3,1.9,.1)
seasonal<-rep(seasfactor,ceiling(n/4))[1:n]
y[1]=e[1];alpha[1]<-5+u[1]
for(t in 2:n){
  y[t]<-seasonal[t]*(alpha[t-1]+e[t])
  alpha[t]<-alpha[t-1]+u[t]
}
plot(y,type="l",main="y and alpha")
  lines(alpha,type="l",lty=3)
```



The seasonal component is evident but is different from the one shown above. In this case, it tends to amplify when the series increases. Indeed, the level α_t is multiplied (not added) to the factor.

Here we can remove the seasonal component by using again the moving average appraoch but in a different way. These are the steps:

Take the centered moving average of the series, call it CMA_t

Divide the CMA from the original series $residuals_t = \frac{y_t}{CMA_t}$

Average the elements of $residuals_t$ by season and obtain the seasonal factor

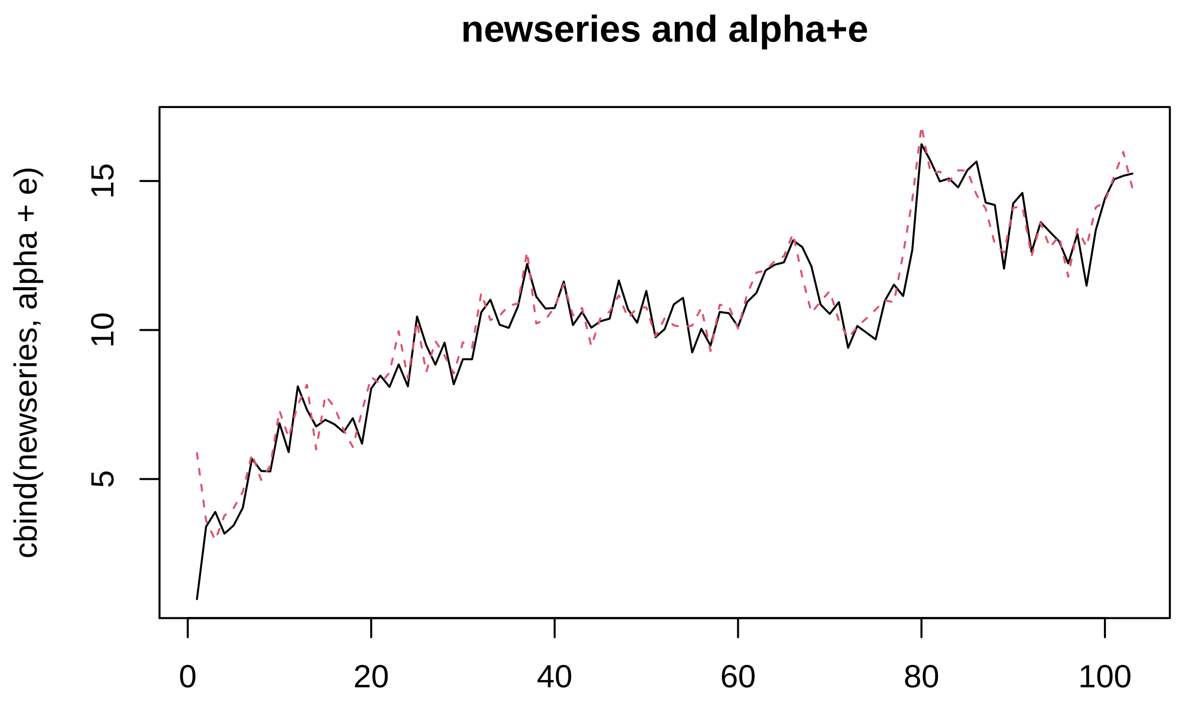
Divide the element of y_t by the corresponding seasonal factor

This is the code (similar to the one above) that makes the trick:

```r
s<-4
n<-length(y)
w<-rep(1/(2*s),s+1);w[2:s]<-1/s
cma<-matrix(NA,length(y),1);
for(g in 1:(length(y)-s)){cma[g+s/2]<-sum(w*y[g:(g+s)])};
residuals<-y/cma
sfactors<-c();for(seas in 1:s){
sfactors[seas]<-mean(na.omit(residuals[seq(seas,length(y)-s+seas,by=s)]))}
sfactors<-sfactors*4/sum(sfactors)
newseries<-y/rep(sfactors,ceiling(n/s))[1:n]
```

Let see how it works with the (multiplicative) seasonal series generate above:

```r
set.seed(7)
n<-103
s<-4
e<-sqrt(.5)*rnorm(n)
u<-sqrt(.4)*rnorm(n)
y<-alpha<-c()
factor<-c(1.7,.3,1.9,.1)
seasonal<-rep(factor,ceiling(n/s))[1:n]
y[1]=e[1];alpha[1]<-5+u[1]
for(t in 2:n){
  y[t]<-seasonal[t]*(alpha[t-1]+e[t])
  alpha[t]<-alpha[t-1]+u[t]
}
#Below I extract the seasonal component
w<-rep(1/(2*s),s+1);w[2:s]<-1/s
cma<-matrix(NA,length(y),1);
for(g in 1:(length(y)-s)){cma[g+s/2]<-sum(w*y[g:(g+s)])};
residuals<-y/cma
sfactors<-c();for(seas in 1:s){
sfactors[seas]<-mean(na.omit(residuals[seq(seas,length(y)-s+seas,by=s)]))}
sfactors<-sfactors*4/sum(sfactors)
newseries<-y/rep(sfactors,ceiling(n/s))[1:n]
matplot(cbind(newseries,alpha+e),type="l",main="newseries and alpha+e")
```



```
factor
```

```
## [1] 1.7 0.3 1.9 0.1
```

```
sfactors
```

```
## [1] 1.66844405 0.30063271 1.93157143 0.09935181
```

We can see that the difference is rather small.

Exercise: create a function that take (1) the series (2) the frequency s (3) the type of seasonality (additive or multiplic) and return the deseasonalised series. Test your code using a real series from the link https://www.eia.gov/totalenergy/data/monthly/index.php.

Seasonal-state-Space representation

Another way to treat the seasonality is to modify the state-space model by taking into consideration the seasonal component. For example, consider the Local level model in the SSOE framework with a seasonal behavior as follows:

$$y_t = \alpha_{t-s} + e_t$$
$$\alpha_t = \alpha_{t-s} + \gamma e_t$$

Where s represents the frequency of the data considered (e.g. monthly, quarterly, weekly, etc.). For example, suppose we observe a quarterly time series following a specific dynamics (say the local level). Moreover, we also observe that, every specific quarter, the time series tends to assume a value being similar to the one of the same quarter last year. We can represent this model as follows:

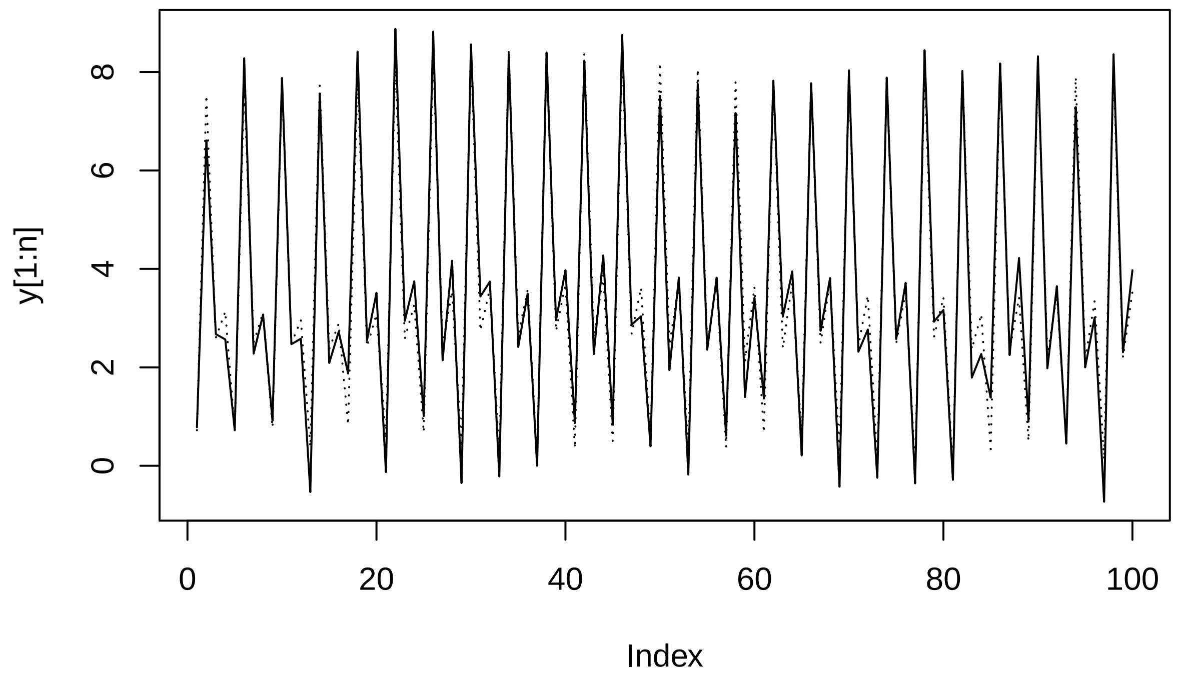
$$y_t = \alpha_{t-4} + e_t$$
$$\alpha_t = \alpha_{t-4} + \gamma e_t$$

For example suppose we want to generate the following quarterly local level model:

where $e \sim Normal(\mu = 0; \sigma_e^2 = .4)$ and $gamma = .3$ where $n = 100$.

```
set.seed(55)
n<-100
e<-sqrt(.4)*rnorm(n)
y<-alpha<-c()
s<-4
sfactor<-matrix(10*runif(s),s,1)
y[1]=sfactor[1]+e[1];
y[2]=sfactor[2]+e[2];
y[3]=sfactor[3]+e[3];
y[4]=sfactor[4]+e[4];
alpha[1]<-sfactor[1]+.2*e[1]
alpha[2]<-sfactor[2]+.2*e[2]
alpha[3]<-sfactor[3]+.2*e[3]
alpha[4]<-sfactor[4]+.2*e[4]
for(t in 5:n){
  alpha[t]<-alpha[t-s]+.3*e[t]
  y[t]<-alpha[t-s]+e[t]

}
plot(y[1:n],type="l")
lines(alpha[1:n],type="l",lty=3)
```



The good news is that the same results as shown above apply since we are assuming that the same factors appear every quarter. Therefore we can estimate the model as follows:

```
s<-4
state<-e<-matrix(0,n,1)
state[1:s]<-y[1:s]
logLikConc<-function(myparam){gamma<-abs(myparam);
for (t in (s+1):(n)) {
  e[t]<-y[t]-state[t-s]
  state[t] <-state[t-s]+gamma*e[t]
}
sum(e[2:n]^2)/(n-1)}
myresults<-optim(.4,method = "Brent", lower = 0, upper = 1,fn=logLikConc)
print("this is the estimated gamma")
```

```
## [1] "this is the estimated gamma"
```

```
myresults[[1]]
```

```
## [1] 0.3252955
```

```
print("this is the estimated variance of e")
```

```
## [1] "this is the estimated variance of e"
```

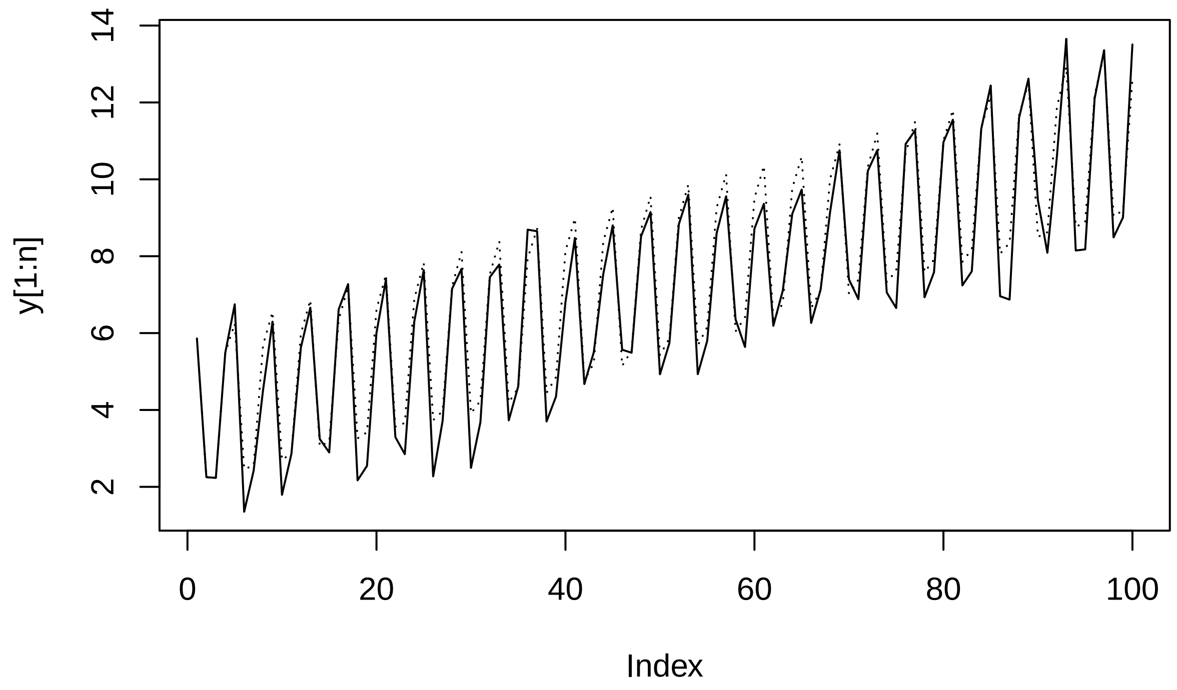
```
myresults[[2]]
```

```
## [1] 0.395181
```

Where 0.325 is the estimated signal-to-noise ratio.

Suppose we want to generate a local level with drift (the model underlying the Theta method).

```
set.seed(1132)
n<-100
e<-sqrt(.4)*rnorm(n)
y<-alpha<-c()
s<-4
co<-.3
sfactor<-matrix(10*runif(s),s,1)
y[1]=sfactor[1]+e[1];
y[2]=sfactor[2]+e[2];
y[3]=sfactor[3]+e[3];
y[4]=sfactor[4]+e[4];
alpha[1]<-sfactor[1]+e[1]
alpha[2]<-sfactor[2]+e[2]
alpha[3]<-sfactor[3]+e[3]
alpha[4]<-sfactor[4]+e[4]
for(t in 5:n){
  alpha[t]<-co+alpha[t-s]+.1*e[t]
  y[t]<-alpha[t-s]+e[t]
  }
plot(y[1:n],type="l")
lines(alpha[1:n],type="l",lty=3)
```



We can now estimate the model as follows:

```r
s<-4
state<-v<-matrix(0,n,1)
state[1:s]<-y[1:s]
logLikConc<-function(myparam){gamma<-abs(myparam[1]);co<-abs(myparam[2]);
for (t in (s+1):(n)) {
  v[t]<-y[t]-state[t-s]
  state[t] <-co+state[t-s]+gamma*v[t]
}
sum(v[2:n]^2)/(n-1)}
myresults<-optim(c(.2,.2),logLikConc)
```

Forecasting time series

We have been discussing the estimation of state-space models but we did not discuss a crucial issue: how can we forecast time series?

One of the most important use of state-space model is to predict time series outside the estimation sample.

A crucial role is played by the state variable α_t. Indeed, the state variable (the unobserved component) is the key to predict our data h-steps ahead.

Consider again model (1) and suppose we want to forecast the y_t variable $1, 2, \cdots, h$-steps ahead. Then we have:

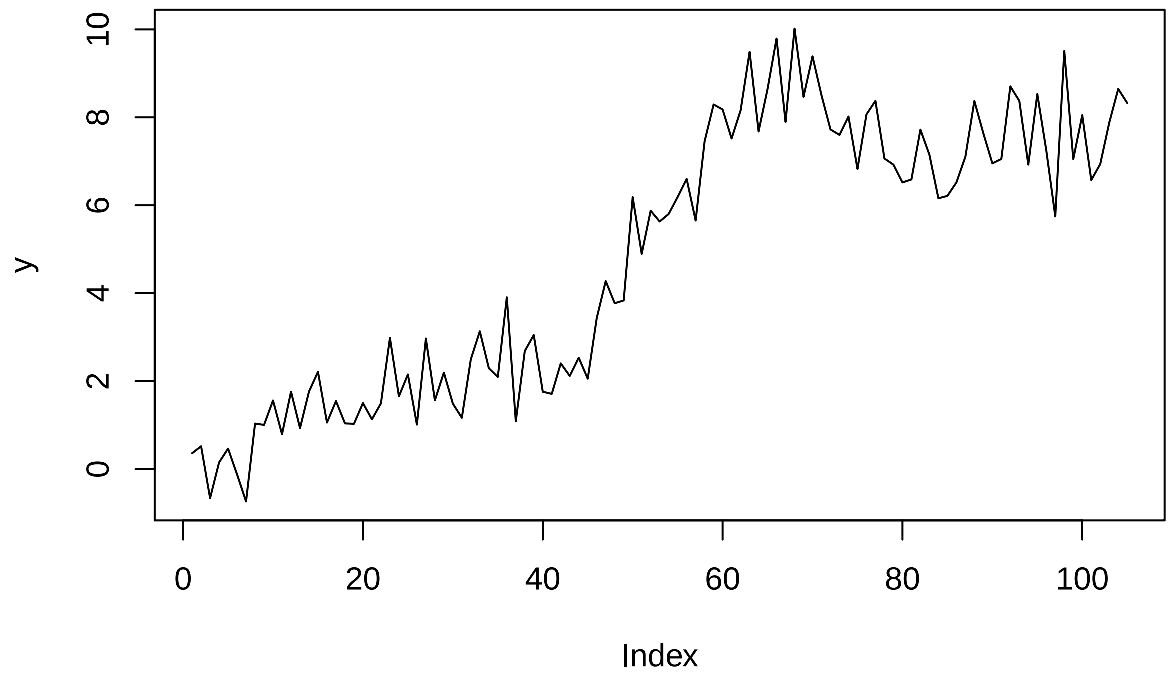
$$
\begin{aligned}
\hat{y}_{t+1} &= za_t = z(c + wa_{t-1} + kv_t) \\
\hat{y}_{t+2} = z * a_{t+1} &= z * (c + w * a_t) = z * c + w * \hat{y}_{t+1} \\
\hat{y}_{t+3} = z * a_{t+2} &= z * (c + w * a_{t+1}) = z * c + w * \hat{y}_{t+2} \\
\hat{y}_{t+4} = z * a_{t+3} &= z * (c + w * a_{t+2}) = z * c + w * \hat{y}_{t+3} \\
&\vdots \\
\hat{y}_{t+h} &= z * c + w * \hat{y}_{t+h-1}
\end{aligned}
\tag{11}
$$

Note indeed that we know v_t but we do not know v_{t+1} as well as v_{t+2} etc.

For example consider the following Theta method:

```r
set.seed(1347)
n<-105
e<-sqrt(.5)*rnorm(n)
u<-sqrt(.1)*rnorm(n)
y<-alpha<-c()
co<-.06
y[1]=e[1];alpha[1]<-u[1]
for(t in 2:n){
  alpha[t]<-co+alpha[t-1]+u[t]
  y[t]<-alpha[t-1]+e[t]

}
plot(y,type="l")
```



Suppose we know the first 100 observations and we do not know the last 5 observations. If we want to forecast them we need first to estimate the parameters and then run the forecasts.

A simple code is the following:

```
#I define x the variable with 100 observations.
n<-100
x<-y[1:n]
a<-c();p<-c()
a[1]=x[1];p[1]<-10000;
k<-v<-c()
v[1]<-0
funcTheta<-function(parameters){q<-abs(parameters[1]);co<-abs(parameters[2]);
  z<-w<-1
  likelihood<-sigmae<-0
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-x[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
  likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
}
  likelihood+.5*n*log(sigmae/n)
}
results<-optim(c(.6,.2),funcTheta)
q<-abs(results[[1]][[1]]);co<-abs(results[[1]][[2]]);
```

```r
  z<-w<-1
      sigmae<-0
for(t in 2:n){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-x[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+v[t]^2/(z^2*p[t]+1)
  }

#This is the drift parameter
      co
```

```
## [1] 0.08032852
```

```r
#This is the variance of e
    sigmae/(n-1)
```

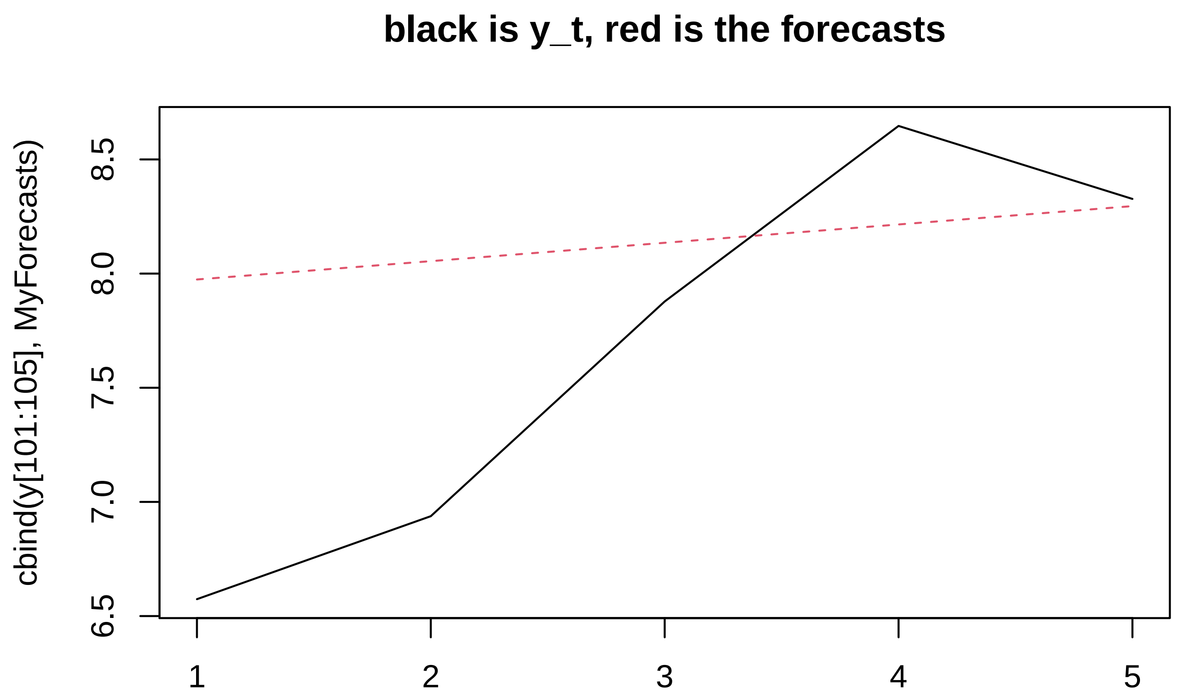
```
## [1] 0.4920203
```

```r
#This is the variance of u
  q*(sigmae/(n-1))
```

```
## [1] 0.1308121
```

Now we can forecast x 5-steps ahead as follows:

```r
MyForecasts<-c()
#This is my one-step ahead for x:
MyForecasts[1]=a[n]
MyForecasts[2]=co+MyForecasts[1]
MyForecasts[3]=co+MyForecasts[2]
MyForecasts[4]=co+MyForecasts[3]
MyForecasts[5]=co+MyForecasts[4]
matplot(cbind(y[101:105],MyForecasts),type="l",main="black is y_t, red is the forecasts")
```



Exercise Forecasting

Generate model (1) using a set of parameters of your choice but using n=107.

Then estimate the model using only the first 100 observations.

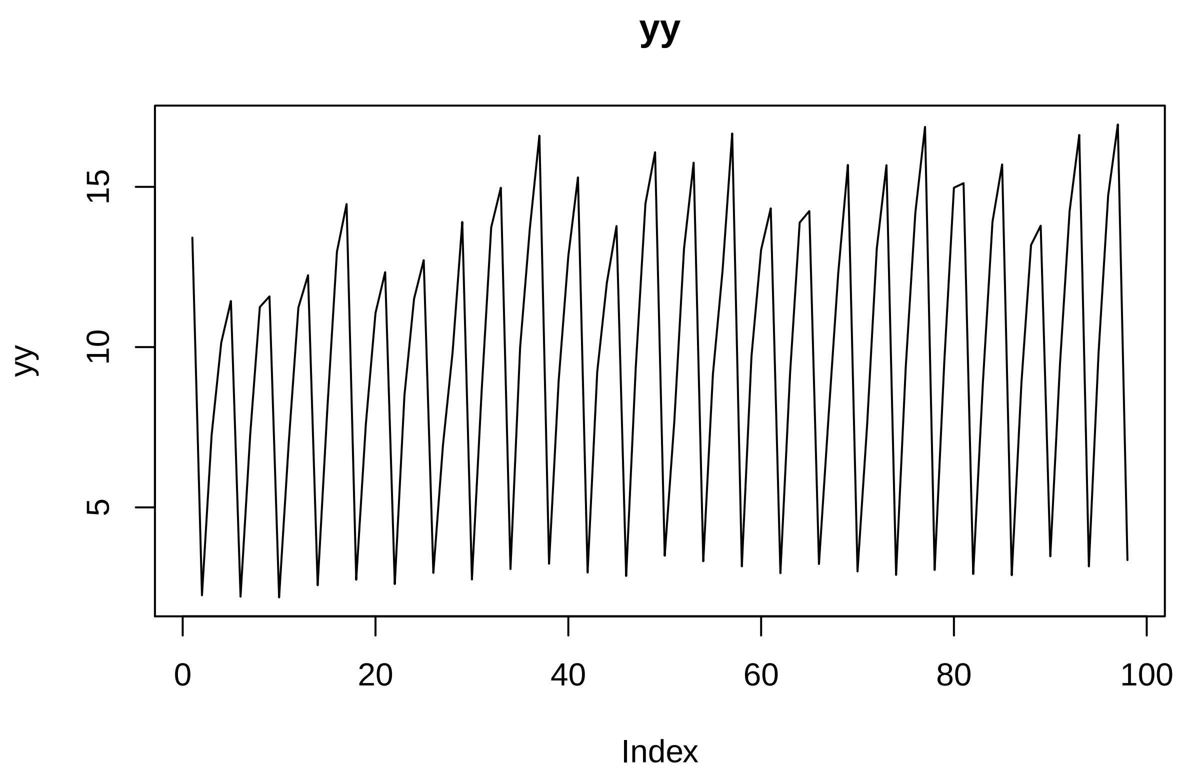
Maky the forecasts for the 7-steps ahead

Forecasting seasonal series

Now consider the case of series affected by a seasonal factor. Suppose we wish to forecast the last 6 observations of this series (considering that we know only the first 100 observations):

```r
set.seed(4241)
n<-1293
h<-6
e<-sqrt(.4)*rnorm(n)
u<-sqrt(.1)*rnorm(n)
my<-alpha<-matrix(NA,n,1)
con<-.03
factor<-c(.3,.9,1.3,1.5)
seasonal<-rep(factor,ceiling(n/4))[1:length(my)]
my[1]=e[1];alpha[1]<-u[1]
```

```r
for(t in 2:n){
  my[t]<-seasonal[t]*(alpha[t-1]+e[t])
  alpha[t]<-con+alpha[t-1]+u[t]
}
yy<-my[300:397]
plot(yy,type="l",main="yy",ylab = "yy")
```



For this type of series the forecasting procedure goes like this: (1) Apply the additive or multiplicative seasonal adjustment procedure discussed above (2) forecast the deseasonalised series with the method you wish (3) add or multiply the seasonal factors to the forecast values obtained.

Remember that the additive or multiplicative procedure depends upon the type of seasonality we observe. If the seasonal component is amplified when the level of the series increase we use the multiplicative procedure. On the other hand, if the seasonal component is constant and is not amplified when the series increases then use the additive procedure.

In this example we suppose we know the first 100 observations and we leave the other 6 as forecasting sample.

```r
 y<-yy[1:(length(yy)-h)]
 s<-4
 n<-length(y)
w<-rep(1/(2*s),s+1);w[2:s]<-1/s
cma<-matrix(NA,length(y),1);
for(g in 1:(length(y)-s)){cma[g+s/2]<-sum(w*y[g:(g+s)])};
residuals<-y/cma
sfactors<-c();for(seas in 1:s){
```

```
sfactors[seas]<-mean(na.omit(residuals[seq(seas,length(y)-s+seas,by=s)]))}
sfactors<-sfactors*s/sum(sfactors)
newseries<-y/rep(sfactors,ceiling(n/s))[1:n]
```

We can now forecast the newly deseasonalised series and then multiply the forecast by the factors to obtain the seasonal forecasts. The code below makes the trick:

```
a<-c();p<-c()
a[1]=newseries[1];p[1]<-10000;
k<-v<-c()
v[1]<-0
funcTheta<-function(parameters){q<-abs(parameters[1]);co<-abs(parameters[2]);
  z<-w<-1
  likelihood<-sigmae<-0
for(t in 2:length(newseries)){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-newseries[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
  likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
}
  likelihood+.5*n*log(sigmae/length(newseries))
}
results<-optim(c(.6,.2),funcTheta)
q<-abs(results[[1]][[1]]);co<-abs(results[[1]][[2]]);
  z<-w<-1
      sigmae<-0
for(t in 2:length(newseries)){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-newseries[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+v[t]^2/(z^2*p[t]+1)
  }

#This is the drift parameter
      co
```

```
## [1] 0.03264525
```

```
#This is the variance of e
    sigmae/(length(newseries)-1)
```

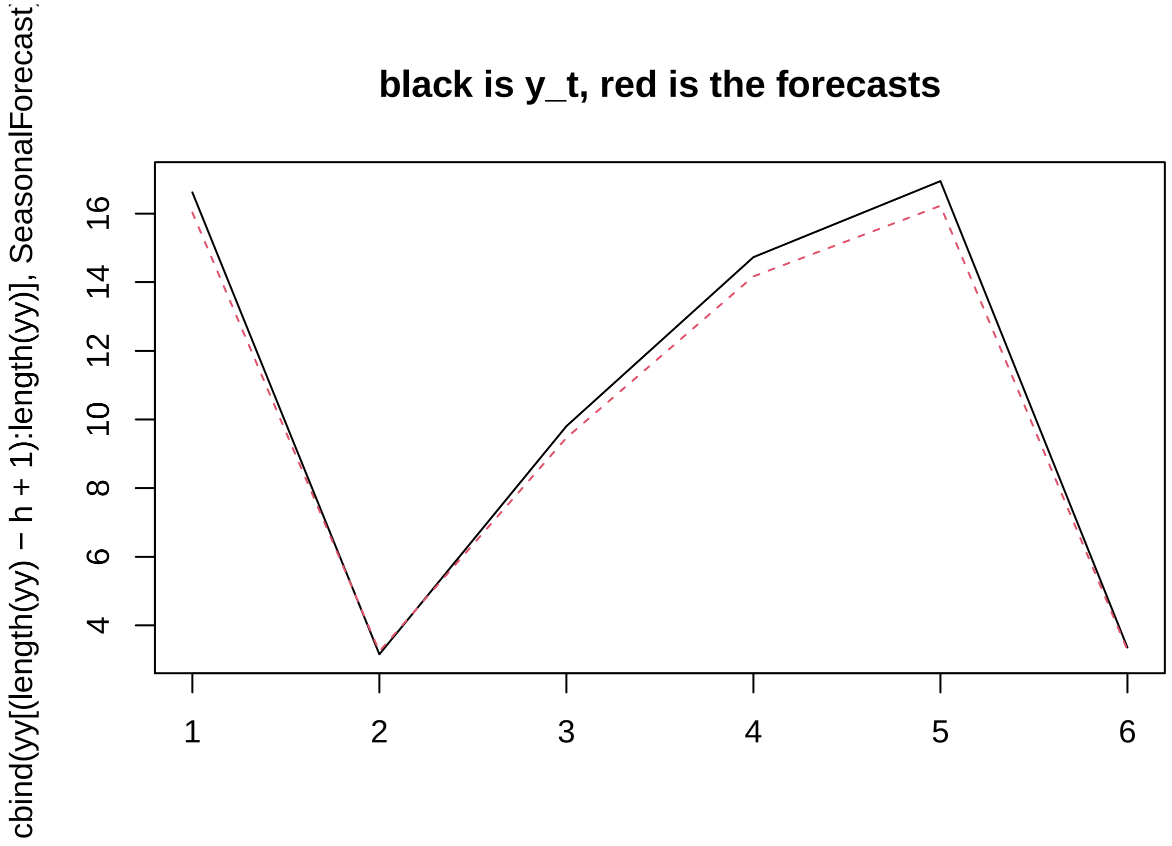
```
## [1] 0.386986
```

```
#This is the variance of u
  q*(sigmae/(length(newseries)-1))
```

```
## [1] 0.05667647
```

Now we can forecast x 6-steps ahead and then multiply the factors to the out-of-sample forecasts, as follows:

```
##This create the out of sample factors to be multiplied
sfactnh<-rep(sfactors,ceiling((n+h)/s))[1:(n+h)]
 sfactout<-sfactnh[(length(sfactnh)-h+1):length(sfactnh)]
w<-z<-1
MyForecasts<-c()
MyForecasts[1]=a[length(newseries)]
for(o in 2:h){
MyForecasts[o]=co+MyForecasts[o-1]
}
SeasonalForecast<-MyForecasts*sfactout
matplot(cbind(yy[(length(yy)-h+1):length(yy)],SeasonalForecast),type="l",main="black is y_t, red is the
```



Unsurprisingly the 6 steps forecasts fit very well the series in black.

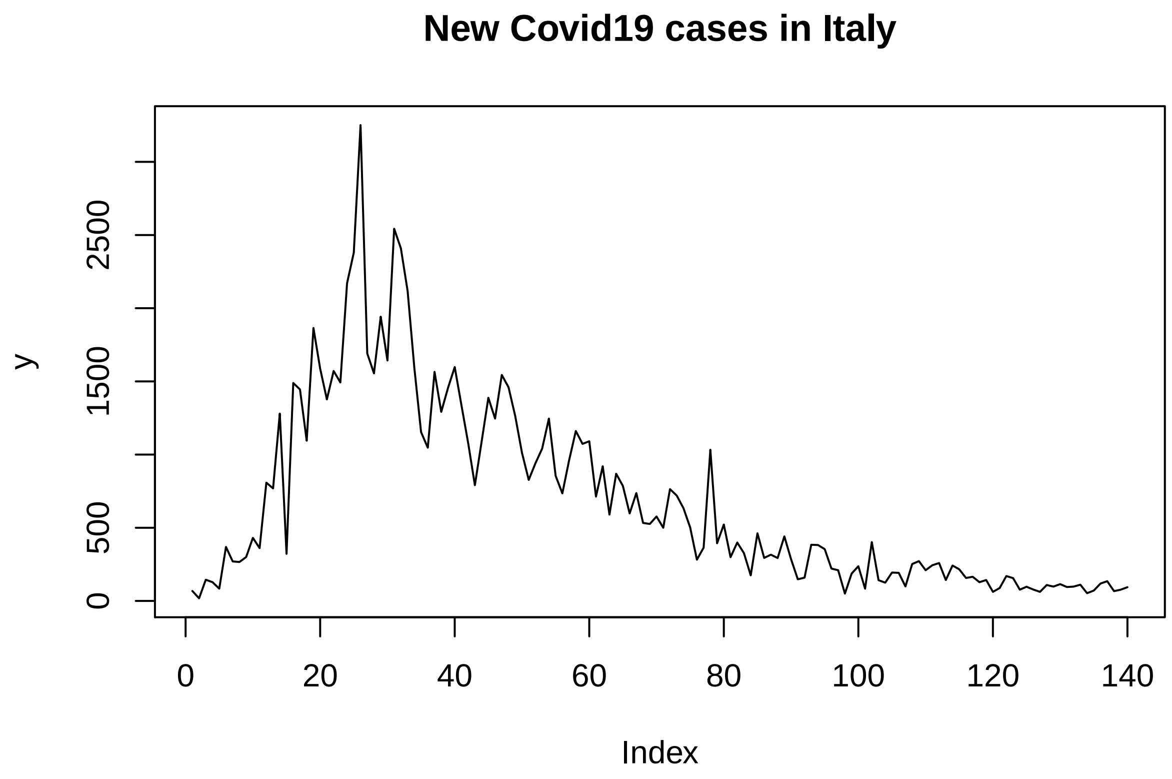
Comparing forecasting performance

Suppose we aim to compare two models when forecasting a specific time series. An important issue is to decide a metric that allow us to establish which model provide better forecasts for the series. The forecasting literature has long discussed the forecast evaluation metrics. Define y_{t+1} y_{t+2} $\cdots$ y_{t+h} the future values of a series (that we do not know) and $\hat{y}_{t+1}$ $\hat{y}_{t+2}$ $\cdots$ $\hat{y}_{t+h}$ the forecasts provided by a specific model. Two popular evaluation metrics are the so called Mean-Absolute-Scaled-Error (MASE) and the Mean-Absolute-Percentage-Error (MAPE) that can be calculated as follows:

$$MASE = \frac{1}{h} \frac{\sum_{t=n+1}^{n+h} |y_t - \hat{y}_t|}{\frac{1}{n-m}\sum_{t=m+1}^{n} |y_t - y_{t-m}|}$$
$$MAPE = \frac{2}{h} \sum_{t=n+1}^{n+h} \frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|} \times 100, \tag{12}$$

Let's make an example. Suppose we want to forecast the number of *newcases* of Coronavirus (Covid19) in the Italian region Lombardy (the most affected region by Covid). Below I provide the code to download data from internet.

```r
#Before running this code you need to install the package "XML"
#Just run install.package("XML")
library("XML")
theURL<-'http://www.pangoo.it/coronavirus/?t=region&r=Lombardia&data=y#table'
thedata<-readHTMLTable(theURL)
y<-diff(as.numeric(thedata[[1]]$V8[1:(length(thedata[[1]]$V8)-1)]))
plot(y,type="l",main = "New Covid19 cases in Italy")
```



Here I estimate the Theta method using the first observations and leaving the last 5 observations to be forecasted:

```r
obs<-length(y)-5
x<-y[1:obs]
a<-c();p<-c()
```

```
a[1]=x[1];p[1]<-10000;
k<-v<-c()
v[1]<-0
funcTheta<-function(parameters){q<-abs(parameters[1]);co<-(parameters[2]);
  z<-w<-1
  likelihood<-sigmae<-0
for(t in 2:obs){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-x[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
  likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
}
  likelihood+.5*n*log(sigmae/length(newseries))
}
results<-optim(c(.6,.2),funcTheta)
q<-abs(results[[1]][[1]]);co<-(results[[1]][[2]]);
  z<-w<-1
      sigmae<-0
for(t in 2:obs){
  k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
  p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
  v[t]<-x[t]-z*a[t-1]
  a[t]<-co+w*a[t-1]+k[t]*v[t]
  sigmae<-sigmae+v[t]^2/(z^2*p[t]+1)
  }

#This is the drift parameter
      co
```

```
## [1] -0.4530988
```

```
#This is the variance of e
    sigmae/(length(newseries)-1)
```

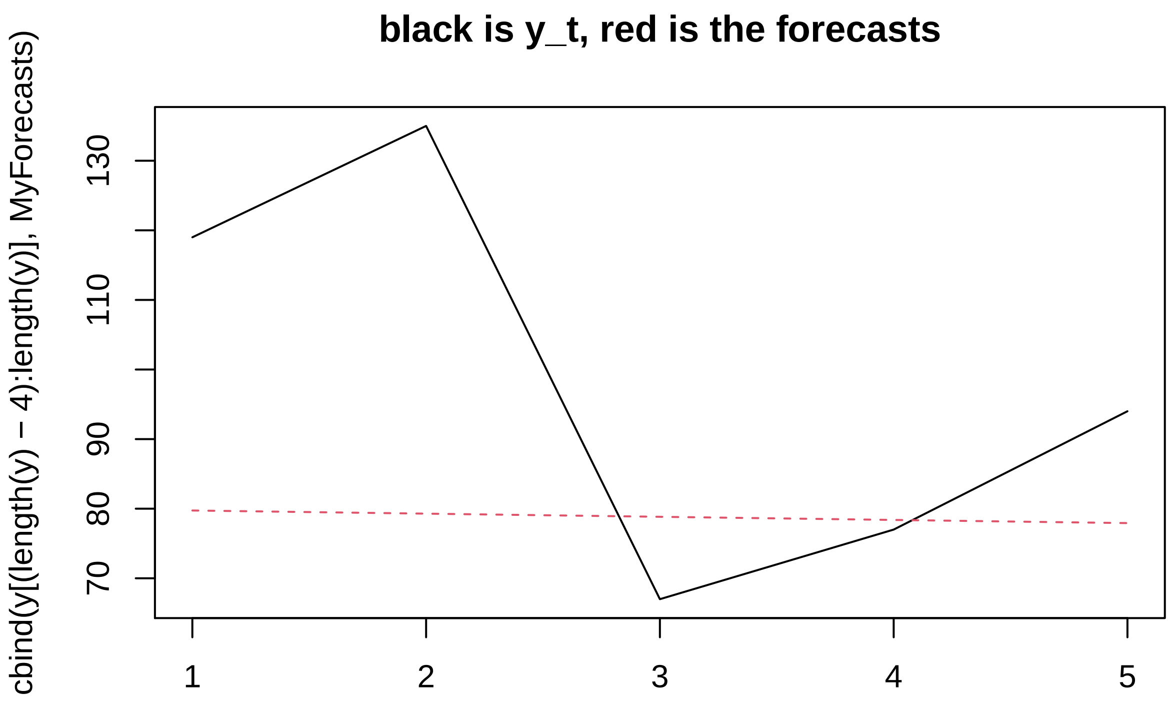
```
## [1] 80674.7
```

```
#This is the variance of u
  q*(sigmae/(length(newseries)-1))
```

```
## [1] 13910.64
```

Here I forecast:

```
MyForecasts<-c()
#This is my one-step ahead for x:
MyForecasts[1]=a[obs]
MyForecasts[2]=co+MyForecasts[1]
MyForecasts[3]=co+MyForecasts[2]
MyForecasts[4]=co+MyForecasts[3]
MyForecasts[5]=co+MyForecasts[4]
matplot(cbind(y[(length(y)-4):length(y)],MyForecasts),type="l",main="black is y_t, red is the forecasts
```



I can now compute the MASE for this method as follows:

```
MASE<-mean(abs(y[(length(y)-4):length(y)]-MyForecasts)/mean(abs(diff(x))))
```

Note in fact that $diff$ provides the first difference of a vector:

```
v<-c(3,1,4,8,2);
diff(v)
```

```
## [1] -2  3  4 -6
```

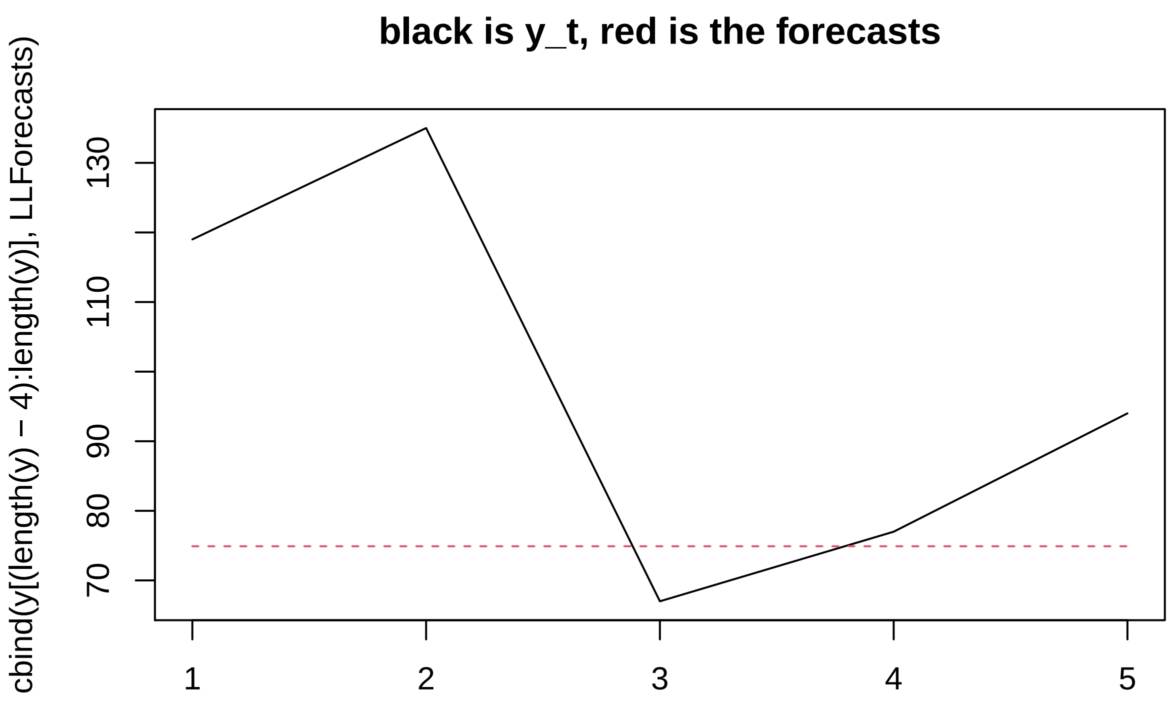
While the sMAPE can be computed as follows:

```
MAPE<-mean(200*abs(y[(length(y)-4):length(y)]-MyForecasts)/(abs(MyForecasts)+abs(y[(length(y)-4):length
```

Now suppose I want to compare the performance with the simple exponential smoothing (Local level model) using the SSOE assumption. I need to estimate and predict the series:

```
a<-v<-matrix(0,obs,1)
a[1]=x[1]
logLikConc<-function(myparam){gamma<-abs(myparam);
w<-z<-1;co<-0;
 for (t in 2:(obs)) {
    v[t]<-x[t]-z*a[t-1]
    a[t] <-co+w*a[t-1]+gamma*v[t]
```

```
  }
  sum(v[2:obs]^2)}
  myresults<-optim(.1,method = "Brent", lower = 0, upper = 1,fn=logLikConc)
  w<-z<-1
a<-v<-matrix(0,obs,1)
a[1]=x[1]
gamma<-myresults[[1]][[1]]
for (t in 2:(obs)) {
    v[t]<-x[t]-z*a[t-1]
    a[t] <-a[t-1]+gamma*v[t]
  }
LLForecasts<-c()
#This is my one-step ahead for x:
LLForecasts[1]=a[obs]
LLForecasts[2]=LLForecasts[1]
LLForecasts[3]=LLForecasts[2]
LLForecasts[4]=LLForecasts[3]
LLForecasts[5]=LLForecasts[4]
matplot(cbind(y[(length(y)-4):length(y)],LLForecasts),type="l",main="black is y_t, red is the forecasts
```



Now if we compare the results of the MASE we have:

```
MASETheta<-mean(abs(y[(length(y)-4):length(y)]-MyForecasts)/mean(abs(diff(x))))
MASELL<-mean(abs(y[(length(y)-4):length(y)]-LLForecasts)/mean(abs(diff(x))))
MASETheta
```

```
## [1] 0.1276149
```

```
MASELL
```

```
## [1] 0.136876
```

Lets check the MAPE:

```
MAPETHETA<-mean(200*abs(y[(length(y)-4):length(y)]-MyForecasts)/
                  (abs(MyForecasts)+abs(y[(length(y)-4):length(y)])))
MAPELL<-mean(200*abs(y[(length(y)-4):length(y)]-LLForecasts)/
               (abs(MyForecasts)+abs(y[(length(y)-4):length(y)])))
MAPETHETA
```

```
## [1] 25.64164
```

```
MAPELL
```

```
## [1] 27.24046
```

Can you now tell which model better forecast the number of new cases of Covid19 in Italy?

Exercise: Forecasting Covid spread in USA regions

The file Excel *CoronavirusSpreadUSAregions.xlsx* contains the daily number of new cases of Covid in five regions of the USA.

Put the data in Rstudio and, for each region, use the first 65 observations for the estimation sample and the last 6 observations for the forecast evaluation.

Moreover, run a forecast comparison and find the model that best predict the virus spread in each region.

Forecast competion in action!

Suppose we have a dataset and we want to compare the forecast performance of different models using the MASE and the MAPE as above.

Below we provide the code for forecasting $h-steps$ ahead using: Model 1 in both the multiple source of error (MSOE) version and the SSOE version. Same for the Theta method in both versions and the damped trend model (SSOE only).

```
ForecastARkf<-function(y,h){

  n<-length(y)
  a<-c();p<-c()
  a[1]=y[1];p[1]<-10000;
  k<-v<-c()
  fu<-function(mypa){q<-abs(mypa[1]);co<-(mypa[2]);w<-1-exp(-abs(mypa[3]))
  likelihood<-sigmae<-0
  z<-1
  for(t in 2:n){
    k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
    p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
    v[t]<-y[t]-z*a[t-1]
    a[t]<-co+w*a[t-1]+k[t]*v[t]
    sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
    likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
  }
  likelihood+.5*n*log(sigmae/n)
  }
  results<-optim(c(.2,1,2),fu)
  v[1]<-0
  z<-1
  q<-abs(results[[1]][[1]]);co<-(results[[1]][[2]]);w<-1-exp(-abs(results[[1]][[3]]))
  sigmae<-0
  for(t in 2:n){
    k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
    p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
    v[t]<-y[t]-z*a[t-1]
    a[t]<-co+w*a[t-1]+k[t]*v[t]
    sigmae<-sigmae+v[t]^2/(z^2*p[t-1]+1)
  }

  Forec<-c()
  Forec[1]<-a[length(y)]
  for(i in 2:h){
    Forec[i]<-co+w*Forec[i-1];
  }
  Forec
}

ForecastAR<-function(y,h){

  state<-v<-matrix(0,length(y),1)
  state[1]=y[1]
  logLikConc<-function(myparam){w<- 1-exp(-abs(myparam[1]));gamma<-abs(myparam[2]);co<-abs(myparam[3])
  for (t in 2:(length(y))) {
    v[t]<-y[t]-state[t-1]
    state[t] <-co+w*state[t-1]+gamma*v[t]
  }
  sum(v[2:length(y)]^2)}
  result<-optim(c(2,.2,1),logLikConc)

  w<-1-exp(-abs(result[[1]][[1]])); gamma<-abs(result[[1]][[2]]); co<-abs(result[[1]][[3]]);

  for (t in 2:(length(y))) {
    v[t]<-y[t]-state[t-1]
    state[t] <-co+w*state[t-1]+gamma*v[t]
  }

  Forec<-c()
  Forec[1]<-state[length(y)]
  for(i in 2:h){
    Forec[i]<-co+w*Forec[i-1];
  }
  Forec
}



ForecastTheta<-function(y,h){

  state<-v<-matrix(0,length(y),1)
  state[1]=y[1]
  logLikConc<-function(myparam){w<-1;gamma<-abs(myparam[1]);co<-abs(myparam[2])
  for (t in 2:(length(y))) {
    v[t]<-y[t]-state[t-1]
    state[t] <-co+w*state[t-1]+gamma*v[t]
  }
  sum(v[2:length(y)]^2)}
  result<-optim(c(.3,1),logLikConc)

  w<-1; gamma<-abs(result[[1]][[1]]); co<-abs(result[[1]][[2]]);

  for (t in 2:(length(y))) {
    v[t]<-y[t]-state[t-1]
    state[t] <-co+w*state[t-1]+gamma*v[t]
  }

  Forec<-c()
  Forec[1]<-state[length(y)]
  for(i in 2:h){
    Forec[i]<-co+w*Forec[i-1];
  }
  Forec
}

ForecastThetakf<-function(y,h){
  n<-length(y)
  a<-c();p<-c()
  a[1]=y[1];p[1]<-10000;
  k<-v<-c()
  v[1]<-0
  funcTheta<-function(parameters){q<-abs(parameters[1]);co<-abs(parameters[2]);
  z<-w<-1
  likelihood<-sigmae<-0
  for(t in 2:n){
    k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
    p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
    v[t]<-y[t]-z*a[t-1]
    a[t]<-co+w*a[t-1]+k[t]*v[t]
    sigmae<-sigmae+(v[t]^2/(z^2*p[t-1]+1))
    likelihood<-likelihood+.5*log(2*pi)+.5+.5*log(z^2*p[t-1]+1)
  }
  likelihood+.5*n*log(sigmae/n)
  }
  results<-optim(c(.3,1),funcTheta)
  q<-abs(results[[1]][[1]]);co<-abs(results[[1]][[2]]);
  z<-w<-1
  for(t in 2:n){
    k[t]<-(z*w*p[t-1])/(z^2*p[t-1]+1)
    p[t]<-w^2*p[t-1]-w*z*k[t]*p[t-1]+q
    v[t]<-y[t]-z*a[t-1]
    a[t]<-co+w*a[t-1]+k[t]*v[t]
  }

  Forecast<-c()
  Forecast[1]<-a[n]
  for(i in 2:h){
    Forecast[i]<-co+Forecast[i-1];
  }
  Forecast
}


ForecastDamped<-function(y,h){

  obs<-length(y)
  damped<-matrix(0,obs,2)
  damped[1,1]=y[1]
  damped[1,2]=0

  inn<-matrix(0,obs,1)


  fmsoe <- function(param){k1<-abs(param[1]);k2<-abs(param[2]);k3<-abs(param[3])

  for (t in 2:(obs)) {
    inn[t]<-y[t]-damped[t-1,1]-k3*damped[t-1,2]
    damped[t,1] <- damped[t-1,1]+k3*damped[t-1,2]+k1*inn[t]
    damped[t,2] <- k3*damped[t-1,2]+k2*inn[t]

  }
  sum(inn[1:obs]^2)/obs}

  result<-optim(c(runif(1),runif(1),runif(1)),fmsoe)

  k1=abs(result[[1]][[1]])
  k2=abs(result[[1]][[2]])
  k3=abs(result[[1]][[3]])
  if(k3>1){k3=1}

  for (t in 2:(obs)) {

    inn[t]<-y[t]-damped[t-1,1]-k3*damped[t-1,2]
    damped[t,1] <- damped[t-1,1]+k3*damped[t-1,2]+k1*inn[t]#(y[t]-holt[t-1,1]-holt[t-1,2])
    damped[t,2] <- k3*damped[t-1,2]+k2*inn[t]#(y[t]-holt[t-1,1]-holt[t-1,2])
  }

  Forecast<-c();
  Forecast[1]<-damped[obs,1]+k3*damped[obs,2]
  for(i in 2:h){
    Forecast[i]<-Forecast[i-1]+damped[obs,2]*k3^i;
  }
  Forecast
}
```

We can now run a real forecast competition evaluating for each model the MASE and sMAPE and their mean and median across the series of the competition. Here we make use of the well known M3-forecast competition as proposed by Spyros Makridakis.

```
#First install.package("Mcomp") then run the following

library("Mcomp")
```

```
## Loading required package: forecast
```

```
MM<-subset(M3,"yearly")
#Here MM contain the list with the set of series used for the competition

#This use the data for the more recent M4 competition

#library(M4comp2018)
#MM<-Filter(function(l) l$period=="Yearly",M4)
```

Below we report the chunck running the competition and showing the results for the mentioned competitors

```
replic<-length(MM)
steps<-6
Method1<-Method2<-Method3<-Method4<-Method5<-Err1<-Err2<-matrix(0,replic,steps)
Err3<-Err4<-Err5<-sErr1<-sErr2<-sErr3<-sErr4<-sErr5<-matrix(0,replic,steps)

for(g in 1:replic){
  y<-MM[[g]][[6]]
  #For the M4 competition use y<-MM[[g]][[2]]
  Method1[g,]<-ForecastAR(y,steps)
  Method2[g,]<-ForecastARkf(y,steps)
  Method3[g,]<-ForecastTheta(y,steps)
  Method4[g,]<-ForecastThetakf(y,steps)
  Method5[g,]<-ForecastDamped(y,steps)

  Err1[g,]<-MM[[g]][[7]]-Method1[g,]
  Err2[g,]<-MM[[g]][[7]]-Method2[g,]
  Err3[g,]<-MM[[g]][[7]]-Method3[g,]
  Err4[g,]<-MM[[g]][[7]]-Method4[g,]
  Err5[g,]<-MM[[g]][[7]]-Method5[g,]

  sErr1[g,]<-Err1[g,]/mean(abs(diff(y)))
```

```r
  sErr2[g,]<-Err2[g,]/mean(abs(diff(y)))
  sErr3[g,]<-Err3[g,]/mean(abs(diff(y)))
  sErr4[g,]<-Err4[g,]/mean(abs(diff(y)))
  sErr5[g,]<-Err5[g,]/mean(abs(diff(y)))
}

ResultsMAPE<-matrix(0,steps,18)

for(s in 1:steps){
  sMAPE<-matrix(0,replic,5);
  for(i in 1:replic){

    sMAPE[i,1]<-mean(200*abs(Err1[i,1:s])/(abs(Method1[g,][1:s])+abs(MM[[g]][[7]][1:s])))
    sMAPE[i,2]<-mean(200*abs(Err2[i,1:s])/(abs(Method2[g,][1:s])+abs(MM[[g]][[7]][1:s])))
    sMAPE[i,3]<-mean(200*abs(Err3[i,1:s])/(abs(Method3[g,][1:s])+abs(MM[[g]][[7]][1:s])))
    sMAPE[i,4]<-mean(200*abs(Err4[i,1:s])/(abs(Method4[g,][1:s])+abs(MM[[g]][[7]][1:s])))
    sMAPE[i,5]<-mean(200*abs(Err5[i,1:s])/(abs(Method5[g,][1:s])+abs(MM[[g]][[7]][1:s])))


  }
  ResultsMAPE[s,1]<-mean(sMAPE[,1])
  ResultsMAPE[s,2]<-mean(sMAPE[,2])
  ResultsMAPE[s,3]<-mean(sMAPE[,3])
  ResultsMAPE[s,4]<-mean(sMAPE[,4])
  ResultsMAPE[s,5]<-mean(sMAPE[,5])
  ResultsMAPE[s,6]<-mean(sMAPE[,1])/mean(sMAPE[,2])
  ResultsMAPE[s,7]<-mean(sMAPE[,1])/mean(sMAPE[,3])
  ResultsMAPE[s,8]<-mean(sMAPE[,1])/mean(sMAPE[,4])
  ResultsMAPE[s,9]<-mean(sMAPE[,1])/mean(sMAPE[,5])
  ResultsMAPE[s,10]<-median(sMAPE[,1])
  ResultsMAPE[s,11]<-median(sMAPE[,2])
  ResultsMAPE[s,12]<-median(sMAPE[,3])
  ResultsMAPE[s,13]<-median(sMAPE[,4])
  ResultsMAPE[s,14]<-median(sMAPE[,5])
  ResultsMAPE[s,15]<-median(sMAPE[,1])/median(sMAPE[,2])
  ResultsMAPE[s,16]<-median(sMAPE[,1])/median(sMAPE[,3])
  ResultsMAPE[s,17]<-median(sMAPE[,1])/median(sMAPE[,4])
  ResultsMAPE[s,18]<-median(sMAPE[,1])/median(sMAPE[,5])

}

ResultsMASE<-matrix(0,steps,18)

for(s in 1:steps){
  sMASE<-matrix(0,replic,5);
  for(i in 1:replic){
    sMASE[i,1]<-mean(abs(sErr1[i,1:s]));
    sMASE[i,2]<-mean(abs(sErr2[i,1:s]));
    sMASE[i,3]<-mean(abs(sErr3[i,1:s]));
    sMASE[i,4]<-mean(abs(sErr4[i,1:s]));
    sMASE[i,5]<-mean(abs(sErr5[i,1:s]));

  }
  ResultsMASE[s,1]<-mean(sMASE[,1])
  ResultsMASE[s,2]<-mean(sMASE[,2])
  ResultsMASE[s,3]<-mean(sMASE[,3])
  ResultsMASE[s,4]<-mean(sMASE[,4])
  ResultsMASE[s,5]<-mean(sMASE[,5])
  ResultsMASE[s,6]<-mean(sMASE[,1])/mean(sMASE[,2])
  ResultsMASE[s,7]<-mean(sMASE[,1])/mean(sMASE[,3])
  ResultsMASE[s,8]<-mean(sMASE[,1])/mean(sMASE[,4])
  ResultsMASE[s,9]<-mean(sMASE[,1])/mean(sMASE[,5])
  ResultsMASE[s,10]<-median(sMASE[,1])
  ResultsMASE[s,11]<-median(sMASE[,2])
  ResultsMASE[s,12]<-median(sMASE[,3])
  ResultsMASE[s,13]<-median(sMASE[,4])
  ResultsMASE[s,14]<-median(sMASE[,5])
  ResultsMASE[s,15]<-median(sMASE[,1])/median(sMASE[,2])
  ResultsMASE[s,16]<-median(sMASE[,1])/median(sMASE[,3])
  ResultsMASE[s,17]<-median(sMASE[,1])/median(sMASE[,4])
  ResultsMASE[s,18]<-median(sMASE[,1])/median(sMASE[,5])

}

round(ResultsMASE,3)
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] [,12]
## [1,] 1.122 1.027 1.099 1.040 1.123 1.093 1.021 1.079 0.999 0.727 0.673 0.747
## [2,] 1.455 1.349 1.408 1.360 1.489 1.079 1.034 1.070 0.977 0.947 0.918 0.944
## [3,] 1.849 1.722 1.776 1.736 1.889 1.074 1.042 1.066 0.979 1.206 1.185 1.200
## [4,] 2.195 2.045 2.103 2.062 2.243 1.073 1.043 1.064 0.979 1.490 1.460 1.479
## [5,] 2.505 2.344 2.397 2.363 2.579 1.069 1.045 1.060 0.971 1.681 1.737 1.717
## [6,] 2.800 2.626 2.677 2.646 2.886 1.066 1.046 1.058 0.970 1.913 1.920 1.905
##      [,13] [,14] [,15] [,16] [,17] [,18]
## [1,] 0.703 0.761 1.081 0.974 1.035 0.956
## [2,] 0.934 1.030 1.031 1.004 1.014 0.919
## [3,] 1.186 1.236 1.018 1.005 1.017 0.976
## [4,] 1.458 1.470 1.021 1.008 1.022 1.014
## [5,] 1.708 1.697 0.968 0.979 0.984 0.990
## [6,] 1.911 1.922 0.996 1.004 1.001 0.995
```

```
round(colMeans(ResultsMASE),3)
```

```
##  [1] 1.988 1.852 1.910 1.868 2.035 1.076 1.038 1.066 0.979 1.327 1.316 1.332
## [13] 1.317 1.353 1.019 0.996 1.012 0.975
```

```
round(ResultsMAPE,3)
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]  [,6]  [,7]  [,8]  [,9]  [,10]  [,11]
## [1,]  7.112  6.253  7.160  6.327  7.514 1.137 0.993 1.124 0.947  3.029  2.840
## [2,]  8.620  7.874  8.524  7.916  9.166 1.095 1.011 1.089 0.940  4.295  3.926
## [3,] 10.923 10.256 10.687 10.268 11.528 1.065 1.022 1.064 0.948  6.300  5.909
## [4,] 13.014 12.451 12.736 12.445 13.810 1.045 1.022 1.046 0.942  7.904  7.817
## [5,] 14.922 14.440 14.539 14.408 15.987 1.033 1.026 1.036 0.933  9.699  9.251
## [6,] 16.729 16.342 16.331 16.300 18.111 1.024 1.024 1.026 0.924 11.299 11.089
##       [,12]  [,13]  [,14] [,15] [,16] [,17] [,18]
## [1,]  3.195  3.106  3.358 1.067 0.948 0.975 0.902
## [2,]  4.281  4.205  4.770 1.094 1.003 1.022 0.901
## [3,]  6.009  6.017  6.446 1.066 1.048 1.047 0.977
## [4,]  7.606  7.599  7.710 1.011 1.039 1.040 1.025
## [5,]  9.272  9.122  9.257 1.049 1.046 1.063 1.048
## [6,] 10.976 11.009 10.685 1.019 1.029 1.026 1.057
```

```
round(colMeans(ResultsMAPE),3)
```

```
##  [1] 11.887 11.269 11.663 11.277 12.686  1.067  1.017  1.064  0.939  7.088
## [11]  6.805  6.890  6.843  7.038  1.051  1.019  1.029  0.985
```

Exercise: Forecast competition with Quarterly data

The M3 competion contains also quarterly data $subset(M3, "quarterly")$**.**

Using the code that allows to extract the seasonal component try to run the forecast competition for quarterly series, using the Forecast codes as above but including the code for treating multiplicative seasonal data